Universitat de Girona
Escola Politècnica Superior

Treball final de grau

Estudi: Grau en Disseny i Desenvolupament de Videojocs

Títol: Colonial: desenvolupament d'un joc d'estratègia marítim amb generació procedural

Document: Memòria

Alumne: Quim Marsal Olivan (u1967081)

Tutor: Gustavo Ariel Patow
Departament: IMAE
Àrea: LSI

Convocatòria: Juny 2023

Índex

1. Introducció

1.1. Propòsit

El gènere dels videojocs d'estratègia és un dels més populars en el mercat. En aquest tipus d'experiència es dóna més importància a la tàctica que a l'acció, i la sort té un paper secundari. En termes generals, el jugador té una perspectiva global sobre l'entorn, s'enfronta a contra un o més oponents (ja siguin controlats pel propi ordinador o altres jugadors), i per avançar ha de planificar curosament les jugades i controlar indirectament les seves unitats a través d'ordres.

Es poden dividir els jocs d'estratègia en dues categories: per torns ("turn-based") i en temps real ("real-time"). Això sí, existeix una àmplia varietat de subgèneres de jocs d'estratègia. Per exemple, en els 4X ("explorar, expandir, explotar i exterminar"), el jugador controla les tropes d'un imperi i l'objectiu és convertir-lo en el més poderós de tots. En els "tower defense", cal col·locar elements de defensa en el camí dels enemics per eliminar-los abans que destrueixin la base. En els d'artilleria, la mecànica principal és la de disparar munició d'alt calibre des de tancs, vaixells i altres aparells de guerra.

Es pot veure que, a causa del paper que interpreta el jugador en el món d'aquest tipus de jocs, acostumen a estar ambientats en escenaris bèl·lics. Aquest és un que sol atreure molt al públic, com es pot veure en la Figura 1. 14 dels 20 jocs més venuts de l'última dècada a Estats Units (Call of Duty, Battlefield, Destiny i Star Wars Battlefront) són shooters. 5 d'ells (Grand Theft Auto V, Red Dead Redemption II, Minecraft, Elder Scrolls V: Skyrim i The Legend of Zelda: Breath of the Wild) són jocs d'acció i aventura. Només 1 (Mario Kart 8) és d'esport.

Top 20 best-selling games from 2010 through 2019 in the U.S.

1. Grand Theft Auto V
2. Call of Duty: Black Ops
3. Call of Duty: Black Ops II
4. Call Of Duty: Modern Warfare 3
5. Call of Duty: Black Ops III
6. Call Of Duty: Ghosts
7. Red Dead Redemption II
8. Call of Duty: WWII
9. Call of Duty: Black Ops IIII
10. Minecraft
11. Call of Duty: Advanced Warfare
12. Call of Duty: Modern Warfare 2019
13. Elder Scrolls V: Skyrim
14. Mario Kart 8
15. Call of Duty: Infinite Warfare
16. Battlefield 1
17. Battlefield 4
18. Destiny
19. The Legend of Zelda: Breath of the Wild
20. Star Wars Battlefront 2015

Figura 1: Els 20 videojocs més venuts de 2010 a 2019 a Estats Units.

Per tant, pot semblar que l'element de combat del gènere d'estratègia hagi de ser suficient per garantir el seu atractiu. Tot i això, aquests videojocs sempre han estat uns dels més intimidadors per a nous jugadors.

Les raons d'aquest fet són vàries, però principalment ho són el peculiar esquema de mecàniques, la vertiginosa profunditat i els tutorials aclaparadors. El jugador mitjà està acostumat a involucrar-se en la lluita de manera directa, i per ell, haver de prendre decisions abans de cada acció és xocant. Necessitarà aprendre a acostumar-se al nou sistema de combat, que és igual o més complex que el que coneix. Per fer-ho, haurà de prendre atenció a tots els missatges d'ajuda que li mostra el joc de forma constant. Aquestes tres característiques combinades, típiques del gènere, són el gran filtre pels jugadors novells.

Per tant, el repte que es va triar pel treball de final de grau és crear un videojoc d'estratègia que no caigués en els mateixos errors. Cal presentar les mecàniques de manera prou amena com perquè qualsevol jugador, independentment del seu nivell d'experiència, pugui gaudir del joc.

1.2. Motivacions

Les motivacions principals rere aquest projecte són essencialment les de posar en pràctica els coneixements que s'han obtingut al llarg dels estudis. Treballar sobre un projecte estructurat que permeti fer-se una idea de la metodologia que segueixen les empreses de desenvolupament de videojoc independents, així com a prendre noves tècniques que ens puguin ser útils de cara a futurs projectes.

1.3. Objectius

L'objectiu d'aquest treball és crear un joc de tipus RTS ("real-time strategy") amb gestió de recursos, construcció d'edificis, navegació marítima i combat. Tindrà elements de shaders, IA, generació procedural i guardat de partida, amb els quals es té poca familiaritat. El joc serà creat amb el motor de Unity i programat amb C#. Sempre que es pugui s'utilitzaran llibreries existents per funcionalitats específiques.

Les tasques a desenvolupar són:

- Buscar recursos atractius
- Ampliar coneixements sobre shaders, IA i algoritmes de generació procedural
- Desenvolupar la mecànica de navegació marítima
- Desenvolupar la mecànica de generació d'illes
- Desenvolupar la mecànica de gestió de recursos
- Desenvolupar la mecànica de control de personatges
- Desenvolupar la mecànica de guardat de partida
- Dissenyar les interfícies
- Afegir partícules, efectes de so i música
- Arrodonir i finalitzar la documentació

1.4. Quadre d'avaluació

Les parts amb més pes del projecte són les mecàniques i la tecnologia. La primera és la que estava marcada des del principi com a prioritat, ja que l'objectiu era crear una experiència sòlida. La segona va acabar tenint una importància inesperada a causa de la dificultat afegida per la generació procedural del món. L'estètica també va demanar certa dedicació per aconseguir un estil coherent en tots els aspectes. Finalment, la narrativa ha tingut un pes no negligible, ja que es va investigar força el període històric en què s'ambienta el joc per evitar incongruències culturals.

Una estimació a la proporció de pesos entre components seria la següent:

Estètica	20 %
Narrativa	10 %
Mecàniques	30 %
Tecnologia	40 %

2. Estudi de viabilitat

Un estudi de viabilitat previ a l'inici del desenvolupament permet valorar si la inversió de temps i diners en el projecte seria encertada. Es tenen en compte els recursos humans, els requeriments de maquinaria i programari i la viabilitat legal. Finalment es realitza un estudi de mercat en el sector per tal d'esbrinar si el projecte tindria cabuda entre l'oferta existent.

2.1. Recursos humans

Per calcular el cost de recursos humans s'han creat dos perfils de treballador i s'ha elaborat una llista de tasques que s'haurien de repartir entre ells. Els salaris associats als perfils s'han calculat amb les dades proveïdes per Stratos, un portal sense ànim de lucre que ofereix notícies, els fòrums i ofertes de feina als membres de la indústria del desenvolupament multimèdia a Espanya. Els perfils són els següents:

- Dissenyador: 25€/hora.
- Programador: 20€/hora.

Les tasques realitzades per cada perfil i la duració estimada de cada una són les següents:

Tasca	**Perfil**	**Hores**	**Cost (€)**
Investigació	Dissenyador	30	750
Planificació	Dissenyador	20	500
Disseny d'interfície d'usuari	Dissenyador	50	1250
Desenvolupament	Programador	600	12000
Proves i optimitzacions	Programador	100	2000
Memòria	Dissenyador	60	1500
Total		**860**	**18000**

Per tant, el cost aproximat del desenvolupament del projecte és de 18.000€.

2.2. Hardware

Per desenvolupar el projecte s'ha utilitzat un ordinador portàtil amb les següents prestacions:

- Processador: AMD Ryzen 7 3700U
- Memòria RAM: 16 GB DDR4
- Targeta gràfica: AMD Radeon(TM) RX Vega 10 Graphics

El cost de l'ordinador en el moment de compra va ser de 700€.

2.3. Software

Tot el software utilitzat al llarg del procés de desenvolupament és gratuït. Per tant, el cost en recursos de software serà nul.

2.4. Viabilitat legal

En quant a drets legals i propietat intel·lectual, tots els recursos externs que hem inclòs en el nostre projecte s'han obtingut de fonts d'ús lliure i, sempre que així s'especifiqui, donant crèdit a l'autor corresponent. Tot el programari utilitzat s'ha descarregat de fonts oficials.

2.5. Estudi de mercat

Per tal de trobar la competència més semblant a la proposta de projecte, s'han determinat els dos grans pilars que formen la proposta: simulació de civilització i navegació. Llavors, s'han buscat exemples de jocs que representin el màxim exponent de cada una de les dues parts. S'han utilitzat les següent paraules claus (en anglès, ja que proporciona més resultats) el motor de cerca de Google:

- Strategy games (jocs d'estratègia)
- RTS games (jocs d'estratègia en temps real)
- Resource management games (jocs de gestió de recursos)
- Simulation games (jocs de simulació)
- Sailing games (jocs de navegació)

Dels resultats obtinguts, els més destacables han estat Age of Empires i Windward.

2.5.1. Age of Empires

Existeixen molts jocs d'estratègia en temps real amb una ambientació similar a la de la proposta (Northguard, Banished, Farthest Frontier, Settlement Survival, etc.). El més destacable és Age of Empires (Figura 2), la famosa saga desenvolupada originalment per Ensemble Studios. En el mode de joc principal, el jugador selecciona una civilització per jugar en un mapa creat a l'atzar. A partir de llavors és qüestió de gestionar els recursos, millorar l'estat defensiu i ofensiu i avançar el nivell tecnològic al llarg de la història. L'ambientació de Age of Empires III, en particular, comença l'any 1421, quan Europa estava colonitzant les Amèriques.

Figura 2: Age of Empires III

2.5.2. Windward

Hi ha diversos jocs centrats en la navegació i el combat naval, tot i que pocs tenen una jugabilitat semblant a la de la proposta. Jocs com Sea of Thieves, Sailwind i Blackwake tenen una perspectiva de primera persona i permeten moure's per l'embarcació com a part de la tripulació, el qual no interessa per aquest projecte. Maelstrom, Naval Action i King of Seas tenen la càmera centrada en el vaixell, però el seu estil de joc no és exactament el que es busca. El primer és molt més frenètic, el segon molt metòdic i el tercer molt narratiu. Windward (Figura 3) és potser el més similar a la idea original, amb una perspectiva aèria i una jugabilitat senzilla però sòlida.

Figura 3: Windward

2.5.3. Quadre comparatiu

Un cop identificats els jocs més exemplars de les dues categories, es comparen segons els aspectes que es consideren més importants. També es fa amb la mateixa proposta, i d'aquesta manera es pot veure quines fortaleses tindrà en comparació als altres.

	Age of Empires	**Windward**	**Colonial**
Gestió de recursos	10	6	8
Simulació	9	5	8
Combat	6	8	7
Estètica	9	6	7

Encara que la proposta no aconsegueix la millor nota en cap aspecte, en tots ells supera en un dels altres dos jocs. És a dir, serà més complet que qualsevol dels dos. Només per aquest fet es considera que la proposta té potencial i val la pena dedicar els recursos necessaris per fer-la realitat.

2.6. Públic objectiu

A l'hora de fer un videojoc és important identificar el públic objectiu a qui va destinat i tenir una idea detallada dels jugadors potencials del joc. Això ens ajuda a dissenyar un joc que s'adapti millor als gustos i preferències d'aquest públic.

En el cas d'aquest projecte, que s'intenta portar el gènere dels jocs d'estratègia a una audiència més àmplia, és contraintuïtiu delimitar el públic objectiu. Tot i això, sí que es pot establir un perfil de jugador ideal.

2.6.1. Tipologia de jugadors

Per definir el perfil de jugador, s'utilitza la classificació presentada per Richard Bartle (veure Figura 4), que segons l'estil de joc de cadascú determina en quin dels següents 4 perfils encaixa millor:

- **Triomfadors**: Interessats a actuar en el món. Tenen com a objectiu resoldre el major nombre de reptes i dificultats per poder demostrar l'èxit en un joc. Normalment inverteixen moltes hores per aconseguir recompenses que els retornin benefici, senzillament pel prestigi de tenir-lo. Són jugadors que requereixen de novetats constants en el joc, ja que sinó un cop obtinguts tots els èxits, abandonen el joc, ja que normalment el contacte amb la comunitat és nul.
- **Exploradors**: Interessats en interactuar amb el món. Prefereixen descobrir àrees, creant mapes i aprenent a descobrir el que està amagat. Sovint se senten restringits quan un joc delimita el temps, i no els permet mirar al voltant al seu propi ritme. Són jugadors exigents i igual que els triomfadors, i sempre demanen d'evolucions i novetats en dintre del joc.
- **Competidors**: Interessats en actuar sobre altres jugadors. Són molt competitius i, tot i que prefereixen jugar en comunitat, el seu interès en el joc és limitat, ja que prefereixen lluitar contra la resta de jugadors tractant-los com a objectes del joc, i busquen la sensació de dominar la partida.
- **Socialitzadors**: Interessats a interactuar amb els jugadors. Per ells, el joc és merament una eina que utilitzen per conèixer altres jugadors dins o fora d'ell. Són els usuaris que normalment usen elements de comunicació com els xats, foros, wikis, per tal de mantenir la comunitat activa.

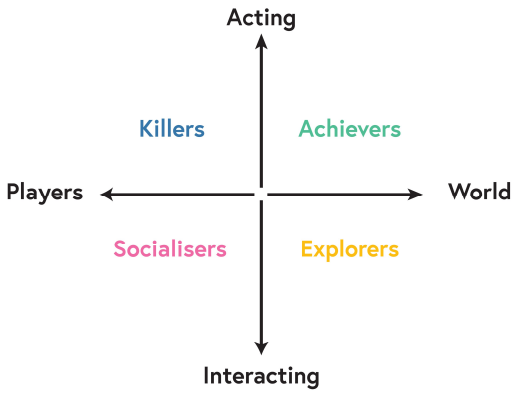


Figura 4: Esquema de la classificació de Bartle

En certa manera, el joc que es planteja està dirigit a tots els perfils excepte els socialitzadors, ja que no compta amb component multijugador. Pels triomfadors hi ha la gestió de recursos i el repta que comporta maximitzar la producció. Pels exploradors hi ha l'enorme espai per navegar i la sensació de progressió relacionada amb omplir el mapa. Pels competidors hi ha l'element de combat amb vaixells enemics, que es complementa amb un feedback instantani molt satisfactori.

2.6.2. Perfil de jugador

El perfil de jugador ideal és una persona jove, entre 15 a 25 anys. Juga a videojocs d'acció de forma casual, i el seu tipus de joc preferit és els shooters. Té poca o gens d'experiència amb els jocs d'estratègia, però que tenen ganes de provar experiències noves. Seguint la classificació de Bartle, pertany principalment a la categoria dels triomfadors.

2.6.3. Motivacions, necessitats i emocions del jugador

L'element impulsor de la conducta és la motivació, i d'altra banda, les emocions condicionen la motivació. Per tant, cal identificar les emocions que es volen provocar en el jugador perquè estigui motivat a jugar.

En el llibre "The Art of Game Design" de Jesse Schell es llisten algunes de les emocions positives que es poden sentir mentre es juga. Aquestes són:

- **Anticipació**: Sentim plaer quan pensem que algun tipus de plaer està per arribar. L'espera és un tipus de plaer en sí mateixa.
- **Gaudir de la desgràcia aliena**: Cert plaer basat en la desgràcia d'un altre. Al voltant d'aquest plaer giren algunes accions dirigides a aconseguir una venjança. És un aspecte important del joc competitiu.
- **Fer regals**: El plaer de donar-li una cosa a algú sense esperar res a canvi pot ser més gratificant que mantenir la possessió d'aquesta.
- **Humor**: Les simples bromes i acudits són una bona font de positivitat.
- **Possibilitat**: El plaer de tenir moltes opcions disponibles.
- **Realització**: Plaer/orgull al aconseguir portar a terme una acció.
- **Purificació**: El plaer d'acabar una tasca discreta del joc. Per exemple, passar d'un nivell a un altre en un joc perquè es complien els requisits per fer-ho.
- **Sorpresa**: Un canvi sobtat en el paradigma és estimulant, independentment de quin.
- **Emoció**: La sensació d'emoció i perill que passa en alguns jocs. Schell ofereix la següent fórmula: "la por menys la mort és igual a diversió".
- **Guanyar davant l'adversitat**: Es diferencia de la realització ja que en aquesta s'han hagut de superar molts obstacles i hi havia poques possibilitats d'aconseguir-ho.
- **Contemplació**: El plaer d'aturar-se, parar de jugar i experimentar el moment.

Per la proposta es volen provocar les sensacions d'anticipació, possibilitat, realització, emoció, guanyar davant l'adversitat i contemplació.

2.6.4. Arc emocional

Les emocions mencionades en el punt anterior es presentarien al llarg del joc en el següent arc emocional: El jugador sentirà anticipació i possibilitat al navegar amb el vaixell, amb la idea que qualsevol illa que es pugui trobar serà nova i única. Quan prengui decisions mentre gestiona els recursos sentirà realització, ja que són petits reptes que requereixen pensar mínimament. Durant un combat naval tindrà la sensació d'emoció i, si és victoriós, també la de guanyar davant l'adversitat. Finalment, s'espera que pugui sentir contemplació al observar l'estètica, escoltar els sons i valorar els sistemes simulats.

3. Planificació

A l'inici del desenvolupament es va planificar el projecte seguint el model Waterfall o cascada. Es van identificar les fases, es van desgranar en tasques i es va establir l'ordre a partir del qual serien treballades. Una fase no es pot començar si l'anterior no s'ha acabat.

Es va preveure una duració mitjana de tasca de dues setmanes. Es va establir un màxim de 31 dies per tasca, ja que es considera que dedicar més d'un mes a una sola significava deixar la resta de banda. Al final es van escurçar unes tasques i allargar unes altres, però sempre respectant els límits.

La planificació de les tasques del projecte és la següent:

Tasca		Data inici	Duració (dies)	Data fi
Generació procedural	Generació de soroll	20/6/2022	6	26/6/2022
	Creació de mesh	26/6/2022	8	4/7/2022
	Càlcul de navegació	4/7/2022	30	3/8/2022
	Elements de les illes	3/8/2022	12	15/8/2022
Navegació	Shading de l'aigua	15/8/2022	8	23/8/2022
	Controls de moviment i combat	23/8/2022	20	12/9/2022
	IA de l'enemic	12/9/2022	12	24/9/2022
	Visualització del mapa i selecció del destí	24/9/2022	16	10/10/2022
Sistema d'edició	Creació de cel·les	10/10/2022	26	5/11/2022
	Construir tancats	5/11/2022	12	17/11/2022
	Col·locar edificis	17/11/2022	4	21/11/2022
Construccions	Sistema d'inventari	21/11/2022	22	13/12/2022
	Gestió dels animals	13/12/2022	20	2/1/2023
	Disseny dels menús	2/1/2023	14	16/1/2023
Personatges	Models i animacions	16/1/2023	3	19/1/2023
	Rutines	19/1/2023	26	14/2/2023
	Tasques	14/2/2023	28	14/3/2023
Altres	Gestió d'esdeveniments	14/3/2023	18	1/4/2023
	Efectes de so i partícules	1/4/2023	10	11/4/2023
	Serialització	11/4/2023	20	1/5/2023
	Documentació	20/6/2022	345	31/5/2023

En la Figura 5 es mostra el diagrama de Gantt resultant de la planificació.

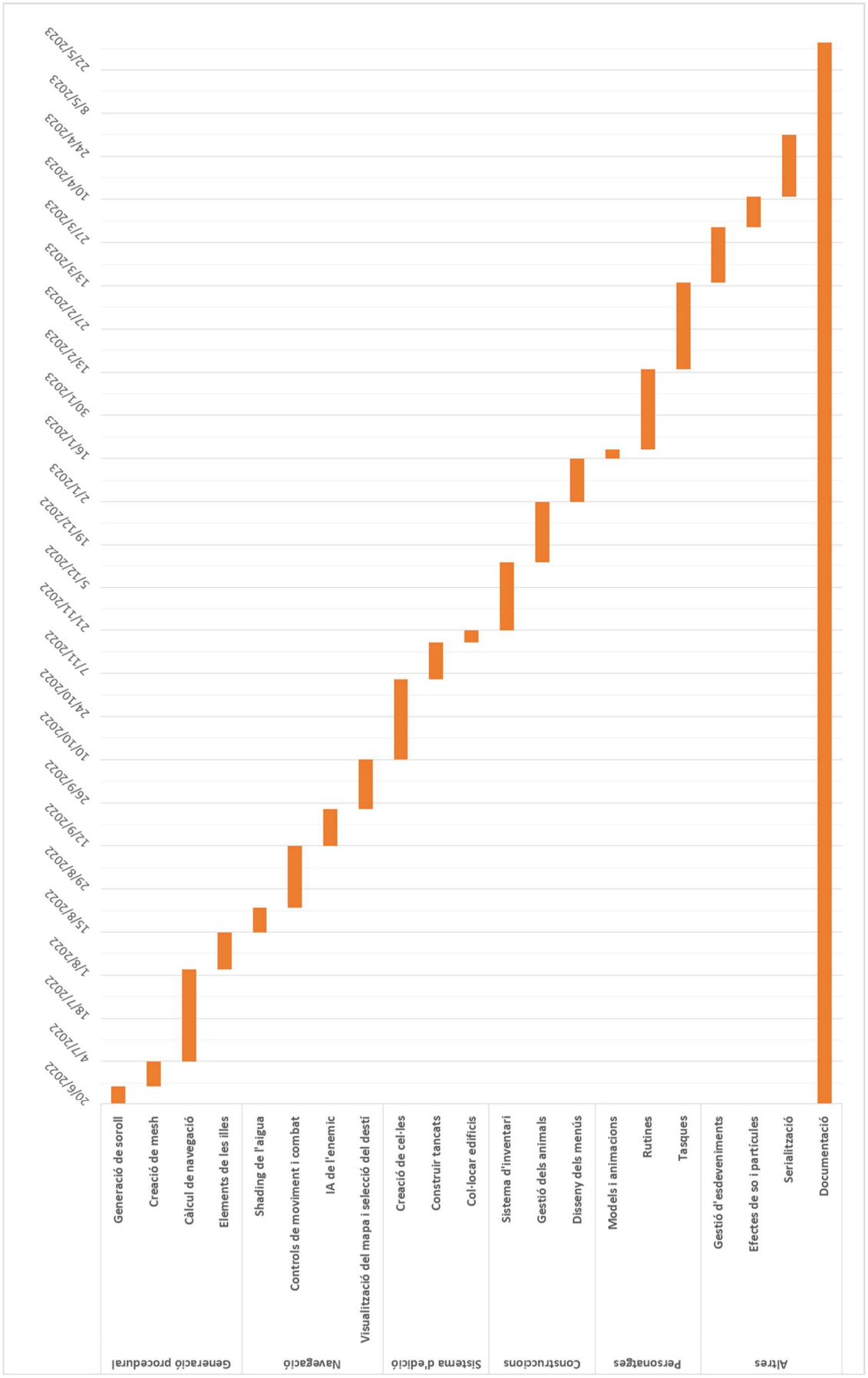


Figura 5: Diagrama de Gantt

4. Marc de treball i conceptes previs

4.1. Conceptes previs

4.1.1. Generació procedural

En informàtica, la generació procedimental és un mètode de creació de dades algorítmicament en lloc de manualment, normalment mitjançant algorismes generats per humans juntament amb l'aleatorietat generada per ordinadors. En els videojocs, s'utilitza per crear automàticament grans quantitats de contingut en un joc.

4.1.2. Soroll

El soroll és una sèrie de nombres aleatoris, normalment disposats en una línia o una quadrícula. Per aplicacions com la generació procedural és essencial l'ús de soroll, que dóna detall en forma d'irregularitat a tot tipus de contingut virtual.

Existeixen diferents tipus de soroll. El soroll blanc (Figura 6) retorna valors aleatoris a cada mostra. Els sorolls coherents com el Worley noise (Figura 7) o el Perlin noise (Figura 8) són graduals, és a dir, cada mostra té certa relació amb les mostres veïnes. Per tant, l'aspecte serà visualment més interessant i més adequat per la generació de textures.

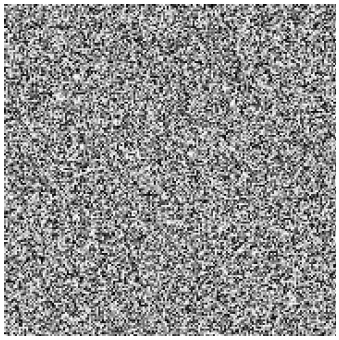


Figura 6: Soroll blanc

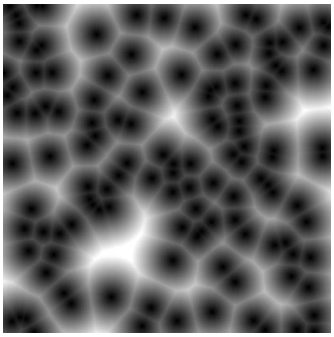


Figura 7: Worley noise

Figura 8: Perlin noise

En l'Annex 1: Soroll en detall s'ha inclòs una exploració més detallada del soroll.

5. Disseny del videojoc

5.1. Idea

El concepte del joc va néixer d'una idea de mecànica que no és molt comuna en els jocs de món obert: que l'espai es defineixi a mesura que el jugador l'explora. D'aquesta forma, l'ordre en el que es descobreixen les zones és consistent amb la visió del desenvolupador però el món creat en cada partida és únic.

Per fer-ho interessant, les parts del món són generades de forma procedural. L'obstacle més gran d'aquesta idea de disseny és que les parts estiguin unides de manera fluida. Per simplificar es va decidir que les parts fossin illes a un mar. Així les vores desiguals poden quedar amagades sota l'aigua. A partir d'aquesta solució i en la idea original es va decidir quina seria l'ambientació del joc: l'Era dels Descobriments (segles XV-XVIII), en què les flotes europees van recórrer el món per conquerir tot el territori encara no descobert.

5.2. Inspiració

La font d'inspiració pel sistema d'edició del terreny és el Clash of Clans (Figura 9). L'espai està distribuït en cel·les quadrades, el qual facilita la col·locació d'elements. A diferència d'aquest joc, però, la construcció dels edificis serà instantània i no requerirà mà d'obra. Això evita molts problemes en el disseny del comportament dels personatges. A més, treu la molèstia d'haver d'esperar, el qual té l'objectiu real d'allargar el joc innecessàriament o aconseguir que els jugadors paguin diners per accelerar el procés. A continuació hi ha una captura del joc.

Figura 9: Clash of Clans

La inspiració principal per la navegació i el combat en vaixell és el Dodko (Figura 10). La UI és bàsicament nul·la, el qual resulta en una experiència de joc molt més calmada. La càmera és llunyana i segueix al jugador sense rotar. Hi ha un mini-mapa a la cantonada indicant la ubicació dels elements més importants com la base o els vaixells enemics. Els vaixells deixen un satisfactori rastre a l'aigua. Totes aquestes decisions de disseny s'adhereixen excepcionalment

a la idea del projecte i s'intentarà captar l'estil en el resultat final. A continuació hi ha una captura del joc.

Figura 10: Dodko

5.3. Característiques del joc

El joc és de gènere RTS (Real Time Strategy). Té una perspectiva 3D i un estil gràfic low-poly. La càmera és aèria amb un angle picat. Depenent de la situació, la posició de la càmera relativa a l'entorn podrà variar. Els controls són els botons del teclat pel moviment del vaixell i de la càmera i el ratolí per la interacció amb la UI i l'entorn.

L'objectiu del joc és explorar el mar en vaixell i descobrir totes les illes. Cada nova illa és cartografiada i s'afegeix a la persistència del món, i sempre que es torni a aquella ubicació hi haurà la mateixa illa.

El repte es trobaria llavors en la gestió de recursos, que servirien per mantenir tant la tripulació com el vaixell en el millor estat possible. Els arbres es poden convertir en fusta i aquesta s'utilitza per construir edificacions o arreglar el vaixell. En el terreny adequat es podran cultivar hortalisses, que alimentaran els habitants de la illa i la tripulació. Fins i tot es poden construir mines d'on s'extrauria pedra i, amb sort, metalls preciosos.

Quan el jugador estigui navegant podrà trobar-se amb vaixells de mercaderies, amb els quals pot comerciar o ser hostil i atacar-los. També es creuarà amb flotes enemigues i combatre-hi. Si guanya el combat aconseguirà tot el seu botí. Si el jugador perd un combat, en canvi, la partida s'haurà acabat.

Tal i com estan plantejades les mecàniques, tot el joc es pot desenvolupar en una sola escena de Unity. Per tant, la transició entre els dos modes de joc serà perfectament seguida.

5.4. Mecàniques

5.4.1. Espai

Com s'ha comentat anteriorment, el joc és de tipus món obert, però tot i que al jugador li pugui semblar que està explorant amb total llibertat, l'ordre amb el que es troba amb els elements de l'entorn està predefinit. És a dir, es tracta d'un món dinàmic construït a partir de les decisions que prengui el jugador, encara que ell no ho sàpiga. Seguint la classificació d'espais en els jocs, l'estructura general del joc tindria una tipologia dinàmica (Figura 11).

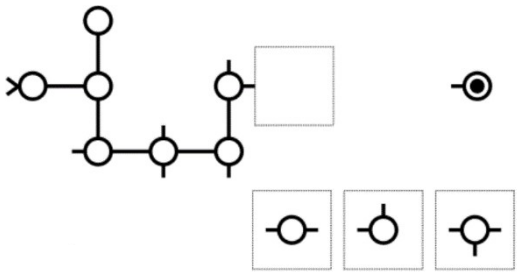


Figura 11: Tipologia d'espai dinàmica

L'estructura del joc es pot explicar amb les següents normes:

- Mentre el jugador està en una illa es dedica a gestionar recursos. Per abandonar una illa ha de clicar un botó i començarà a navegar.

- Mentre el jugador està navegant, si és a prop d'una illa pot entrar-hi clicant un botó. També pot entrar en combat o començar a comerciar o a pescar. Per acabar una sessió de comerç o de pesca només cal clicar un botó. En canvi, el combat s'acabarà quan l'enemic o el jugador morin o fugin un de l'altre.

- El joc s'acaba quan el jugador mor.

Expressada en forma de pseudocodi, seria de la següent manera:

```
En_Illa = fals
Comerciant = fals
En_Combat = fals
Pescant = fals
Mort = fals

Mentre no Mort
    Si En_Illa
        Gestionar recursos

        Si clica botó de marxar d'illa
            En_Illa = fals
        FSi

    Sinó
        Sinó si En_Combat
            Combatre
            Si Estat de Vaixell_Jugador és Destruït
                Mort = cert
            Sinó si Lluny_de_Enemic o Estat de Vaixell_Enemic és
Destruït
                En_Combat = fals
            FSi

        Si Comerciant
            Comerciar
            Si clica botó de deixar de comerciar
                Comerciant = fals
            FSi

        Sinó si Pescant
            Pescar
            Si clica botó de deixar de pescar
                Pescant = fals
            FSi

        Sinó
            Navegar

            Si clica botó de comerciar
                Comerciant = cert

            Sinó si clica botó de pescar
                Pescant = cert

            Sinó Aprop_de_illa i clica botó d'amarrar en illa
                En_Illa = cert
            FSi
        FSi

    FSi
FMentre
```

5.4.2. Reptes i objectius

L'objectiu principal és acumular la major quantitat de recursos. Es poden aconseguir de dues maneres: obtenint-los de les illes o d'altres vaixells, ja sigui a través d'intercanvi o de combat. Aquests mètodes representen les dues parts del joc: l'exploració de les illes i la navegació.

La forma d'aconseguir la major quantitat de recursos de les illes és arribar al màxim nombre d'illes i comptar amb molta tripulació per accelerar les tasques. Per tal de descobrir més illes és necessari navegar, el qual comporta el risc de combat naval, per la qual es necessita tenir el vaixell en bones condicions.

Per poder obtenir recursos de les flotes rivals convé tenir el vaixell en bones condicions per resistir el combat. Això s'aconsegueix reparant-lo després de cada batalla amb fusta, que es pot aconseguir a les illes.

Es podria dir, doncs, que els reptes són de conflicte. Els jocs d'aquest tipus inclouen estratègia, tàctica, logística, supervivència, reducció de forces enemigues i defensa de les unitats vulnerables.

En el diagrama de la Figura 12 es pot veure la jerarquia de reptes.

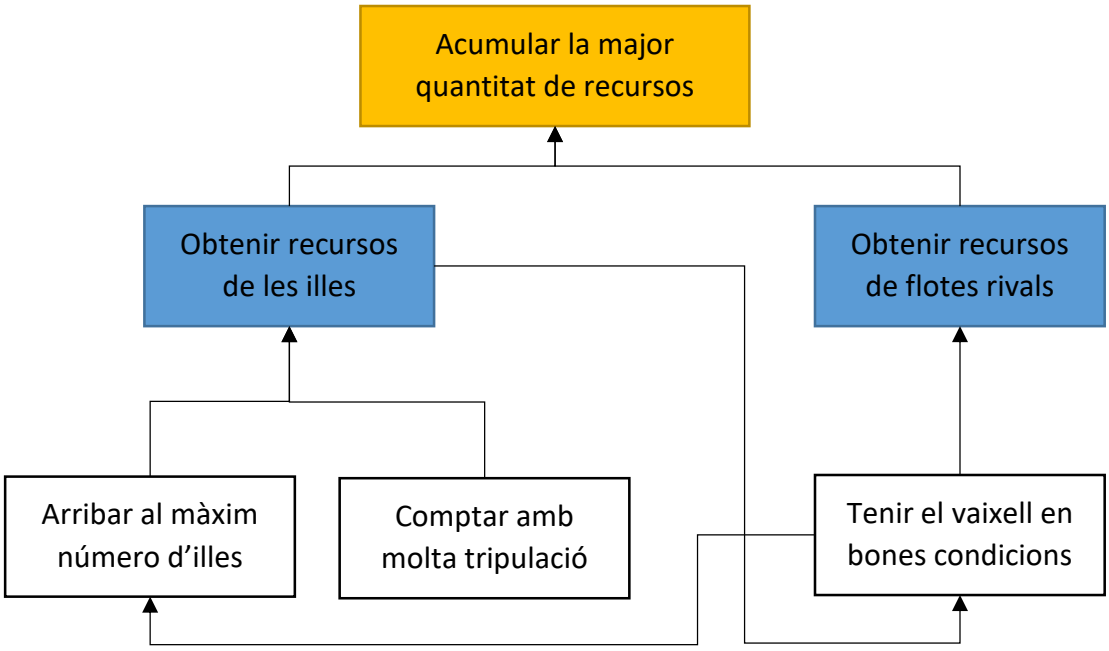


Figura 12: Jerarquia de reptes

5.4.3. Objectes

En aquest punt es llisten els objectes del joc i, si el tenen, es mostra el model 3D. Tots ells són tangibles, ja que ocupen espai en el món ja sigui dins l'inventari, en forma física o ambdós casos.

5.4.3.1. Personatges

Els personatges (Figura 13) inclouen l'home, la dona i el nen.

Figura 13: Models 3D dels personatges. D'esquerra a dreta: home, dona i nen

5.4.3.2. Animals

Els animals (Figura 14) inclouen vedell, vaca, garrí, porc, xai, ovella, pollet i gallina.

Figura 14: Models 3D dels animals. D'esquerra a dreta: vedell, vaca, garrí, porc, xai, ovella, pollet i gallina

5.4.3.3. Elements de l'illa

Els elements de l'illa inclouen arbres (Figura 15), arbustos (Figura 16), roques (Figura 17), flors i bolets (Figura 18).

Figura 15: Models 3D dels arbres

Figura 16: Models 3D dels arbustos

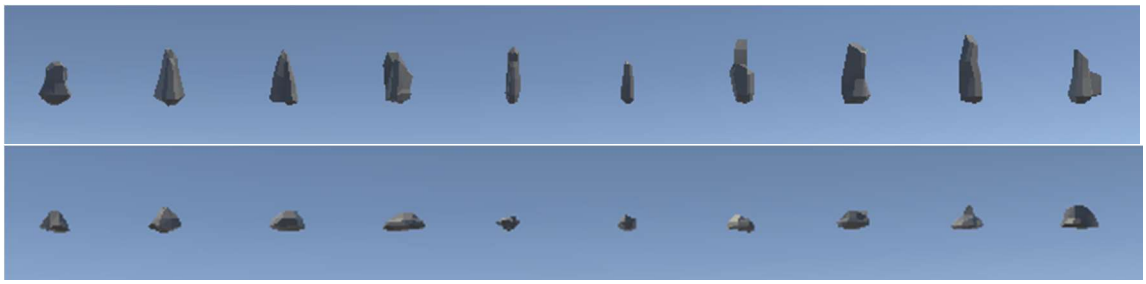


Figura 17: Models 3D de les roques

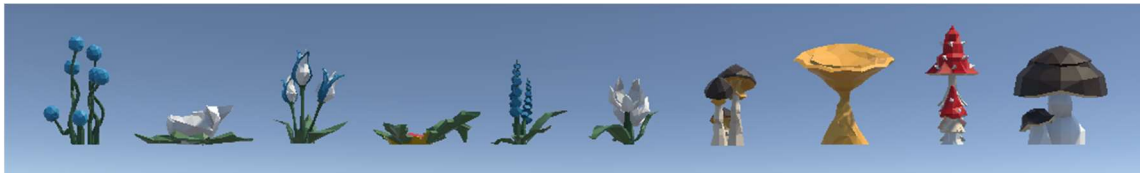


Figura 18: Models 3D de les flors i els bolets

5.4.3.4. Recursos

Els recursos engloben totes les entitats del joc que es poden obtenir, utilitzar, intercanviar, emmagatzemar i transportar. Es divideixen en les següents categories: materials, verdures i carns.

Els materials i les carns no tenen representació física. Els materials provenen dels elements de l'illa i inclouen fusta, pedra, flors, bolets, llavors d'arbre i gemmes. Les carns provenen dels animals i inclouen vedella, carn de porc, carn d'ovella, pollastre i peix.

Hi ha dos tipus de verdures. Les verdures introduïdes pels europeus (Figura 19) són ceba, pastanaga, albergínia, cogombre i enciam.

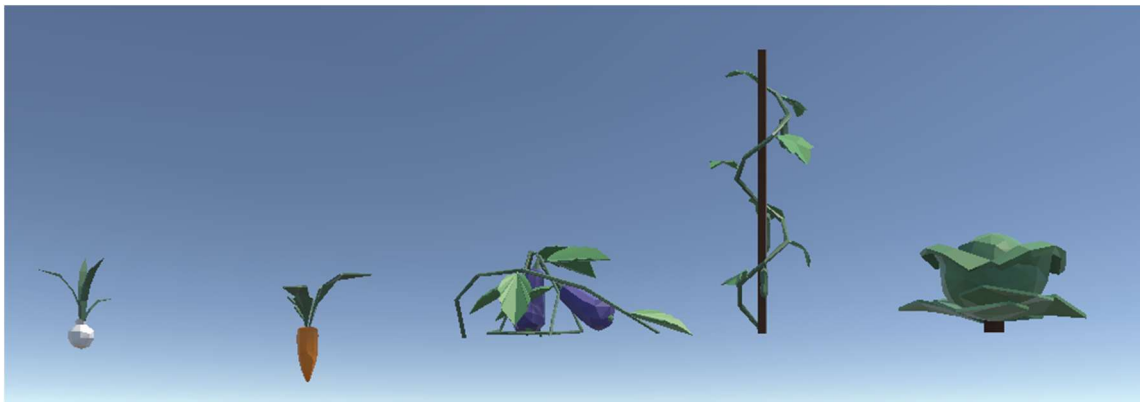


Figura 19: Models 3D de les verdures introduïdes. D'esquerra a dreta: ceba, pastanaga, albergínia, cogombre i enciam

Les verdures natives d'Amèrica (Figura 20) són patata, tomàquet, carbassó, pebrot i blat de moro.

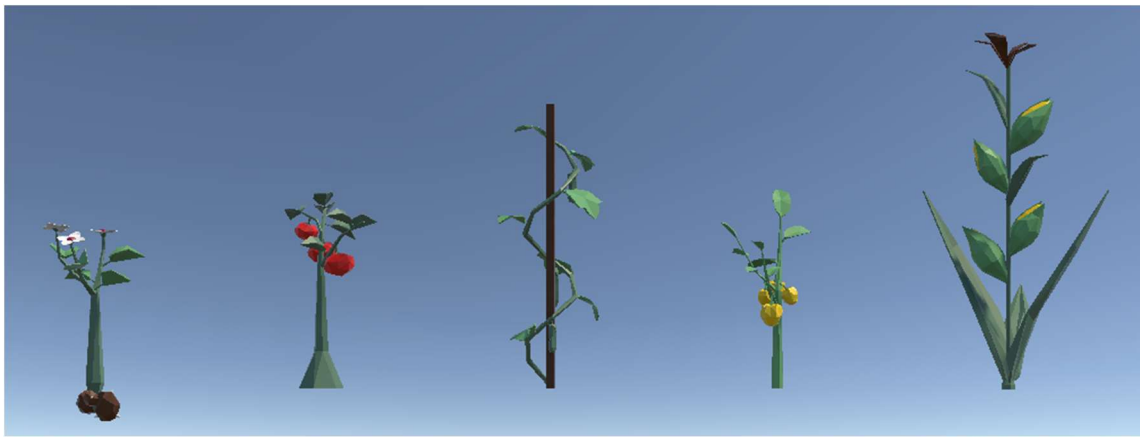


Figura 20: Models 3D de les verdures natives. D'esquerra a dreta: patata, tomàquet, carbassó, pebrot i blat de moro

5.4.3.5. Construccions

Les construccions inclouen tancats, edificis i el vaixell. Quan s'elimina una construcció s'alliberen tots els personatges que hi havia a l'interior, es recuperen els materials de construcció i es deixa el terreny disponible per futures construccions. Cada construcció té un nivell de l'1 al 10 i certs atributs únics que es poden millorar augmentant el nivell. A continuació es presenten tots els tipus de construcció.

5.4.3.5.1. Tancats

- **Horts:** Els horts (Figura 21) permeten cultivar verdures. En cada cel·la de l'interior s'hi adjudica una verdura que, un cop plantada, creixerà i donarà fruits. Cal mantenir les plantes cuidades perquè no es morin. Els personatges es tornaran les plantes a cuidar. Quan es puja de nivell un hort, les plantes creixen més ràpid i tarden més temps a morir.

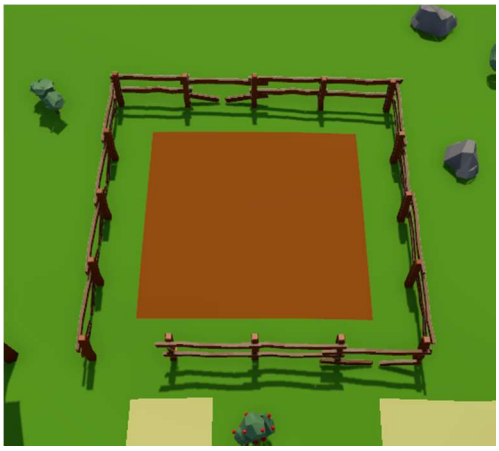


Figura 21: Un hort

- **Corrals**: En un corral (Figura 22) els animals del mateix tipus s'aparelleran i tindran cries. Quan aquestes siguin adultes podran fer el mateix. Un corral té una capacitat màxima, i quan s'hi arribi es sacrificaran animals per fer lloc. Un animal també es sacrificarà quan tingui una edat superior a l'esperança de vida. Quan es puja de nivell un corral, les cries creixen més ràpid i s'obté més carn de cada animal sacrificat. Els personatges no hi poden entrar.

Figura 22: Un corral

5.4.3.5.2. Edificis

- **Cases:** Les cases (Figura 23) permeten als personatges reduir el nivell de cansament. Quan un personatge entra en una casa, estarà uns segons descansant i després en sortirà amb el nivell de cansament al mínim. No es poden enviar personatges a una casa, sinó que hi aniran automàticament quan calgui. Quan es puja de nivell una casa tindrà més capacitat i els personatges tardaran menys a sortir.

Figura 23: Una casa

- **Tavernes:** Les tavernes (Figura 24) tenen un funcionament similar al de les cases, en què els personatges hi entren quan ho necessiten, s'esperen uns segons i en surten automàticament. Les millores quan es puja de nivell també són iguals. En aquest cas, però, es reduirà el nivell de gana.

Figura 24: Una taverna

- **Mines**: Les mines (Figura 25) permeten obtenir pedra i gemmes. Cada cert temps s'extreu una quantitat aleatòria de pedra, que depèn de la quantitat de persones treballant-hi. Hi ha una petita possibilitat d'obtenir una gemma amb cada extracció. Quan es puja de nivell les extraccions s'acceleren i retornen més pedra i gemmes.

Figura 25: Una mina

- **Magatzems**: Els magatzems (Figura 26) serveixen per ampliar la capacitat d'emmagatzematge de l'illa en què es trobin. Un nou magatzem afegit sumarà cert nombre d'espais addicionals en cada una de les categories següents: materials, verdures i carn. Els recursos només ocupen espai de la categoria a la que pertoquen. Aquest és l'únic tipus de construcció que no es pot millorar. En canvi, es pot pujar de nivell de les diferents categories d'inventari de l'illa, el qual augmenta lleugerament la capacitat que cada magatzem hi aporta. Els personatges no hi poden entrar.

Figura 26: Un magatzem

5.4.3.5.3. Vaixell

El vaixell (Figura 27) no està adherit a una illa, sinó que es mou lliurement d'una a una altra. Transporta recursos, animals i tripulació. És la única construcció que no es pot destruir.

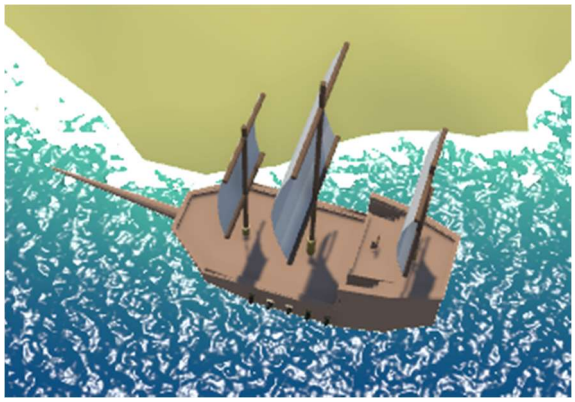


Figura 27: El vaixell

5.4.3.5.4. Taula de diferències

Per aclarir les diferències tècniques entre tots els tipus de construccions s'adjunta la següent taula comparativa.

Construcció		Admet personatges	Permet gestionar els personatges	Es pot millorar	Es pot destruir
Tancat	Hort	X	X	X	X
	Corral			X	X
Edifici	Casa	X		X	X
	Taverna	X		X	X
	Mina	X	X	X	X
	Magatzem				X
Vaixell		X	X	X	

5.4.4. Economia

Els elements del joc tenen una de les 4 funcions possibles dins la pròpia economia:

- **Fonts**: Creen recursos del no res i els guarden en una determinada entitat.
- **Drenatges**: Consumeixen recursos.
- **Convertidors**: Transformen recursos d'un tipus en un altre.
- **Intercanviadors**: Mouen recursos d'una entitat a una altra.

Si les fonts són molt més poderoses que els drenatges el joc serà fàcil perquè el jugador no tindrà gaire incentiu a actuar per obtenir recursos. En canvi, si els drenatges són molt més poderosos que les fonts el joc serà difícil ja que presentarà un repte imponent. Cal balancejar aquests dos per assegurar-se que el joc no sigui avorrit ni aclaparador. Els convertidors i intercanviadors afegiran complexitat al sistema. El jugador haurà de prendre decisions sobre quin recurs gastar per aconseguir-ne un altre. La relació entre les funcions es pot representar amb el diagrama de la Figura 28.

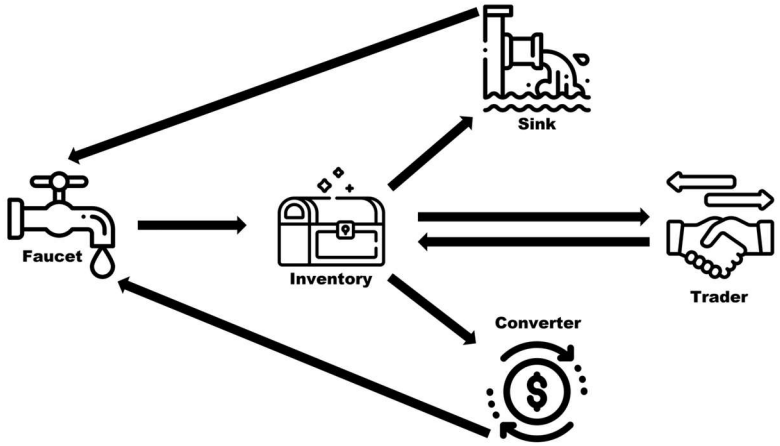


Figura 28: Relació entre les funcions de l'economia

Hi ha unes quantes entitats del joc que es comporten com a fonts. En certa manera, el mar és una font d'illes i de peixos. Les illes són fonts d'elements com arbres i roques. Els horts, els corrals i les mines generen recursos.

El pas del temps té la funció de drenatge en els nivells de gana, cansament i edat dels personatges. Les tavernes, a més, consumeixen menjar per alimentar els personatges. Les bales de canó treuen vida als vaixells i després cal reparar-ho amb fusta.

Totes les construccions requereixen una inversió de materials per ser construïdes. Els elements de les illes també tenen la funció de converter. Per exemple, si es planta un arbre es gasta una llavor però més endavant es podrà obtenir fusta.

Tant les illes com el vaixell poden emmagatzemar recursos, i moure'ls d'una entitat a una altra és portar a terme la funció d'intercanvi. Òbviament, quan el vaixell enemic està obert a comerciar també es comporta com un intercanviador.

5.5. Interfícies

5.5.1. Menús

El menú d'inici del joc es divideix en diverses pantalles. En totes elles apareix el vaixell a la part esquerra i la interfície en la dreta. En la primera pantalla, la d'inici (Figura 29), es mostra el títol del joc i es demana al jugador que cliqui qualsevol tecla per començar.

Figura 29: Pantalla d'inici

En la segona pantalla, la del menú principal (Figura 30), es veuen els botons "Nova partida", "Carregar partida" i "Sortir". Si es clica l'últim es tanca el joc.

Figura 30: Menú principal

Si es clica el de "Nova partida", però, apareix la pantalla per configurar una partida nova (Figura 31). Es demana el nom que tindrà l'arxiu de guardat. Per evitar problemes només s'accepten cadenes alfanumèriques d'un màxim de 15 caràcters. També es demana la llavor que determina l'algorisme de generació d'illes. Serà un nombre d'entre 1 i 5 xifres. Per defecte s'introdueix una llavor aleatòria, que es pot canviar per una escrita per l'usuari o una altra d'aleatòria clicant el botó del dau. Quan els dos valors estan definits es permet clicar el botó "Començar".

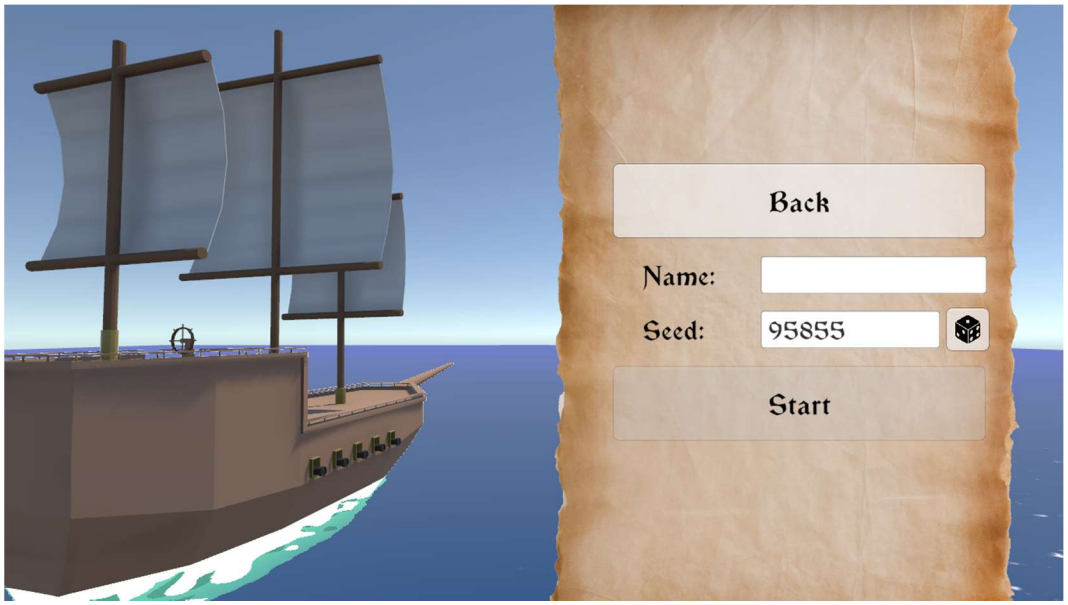


Figura 31: Pantalla de partida nova

Si en canvi es clica el de "Carregar partida", apareix la pantalla per seleccionar una partida existent (Figura 32). Es mostra una llista amb els noms dels arxius que es troben a la carpeta de partides guardades. Per comprovar que la partida es pot carregar es parseja l'arxiu i es busca el valor que indica el temps jugat. Si falla el procés es considera que l'arxiu està corrumput o que no pertany a la carpeta i no es mostra. Tant en aquesta pantalla com en la mencionada anteriorment es pot clicar el botó "Enrere" per tornar al menú principal.

Figura 32: Pantalla de carregar partida

Quan s'inicii una partida nova o es carregui una partida existent es canviarà a l'escena del joc. Per dissimular la transició es mostra una pantalla de càrrega (Figura 33).

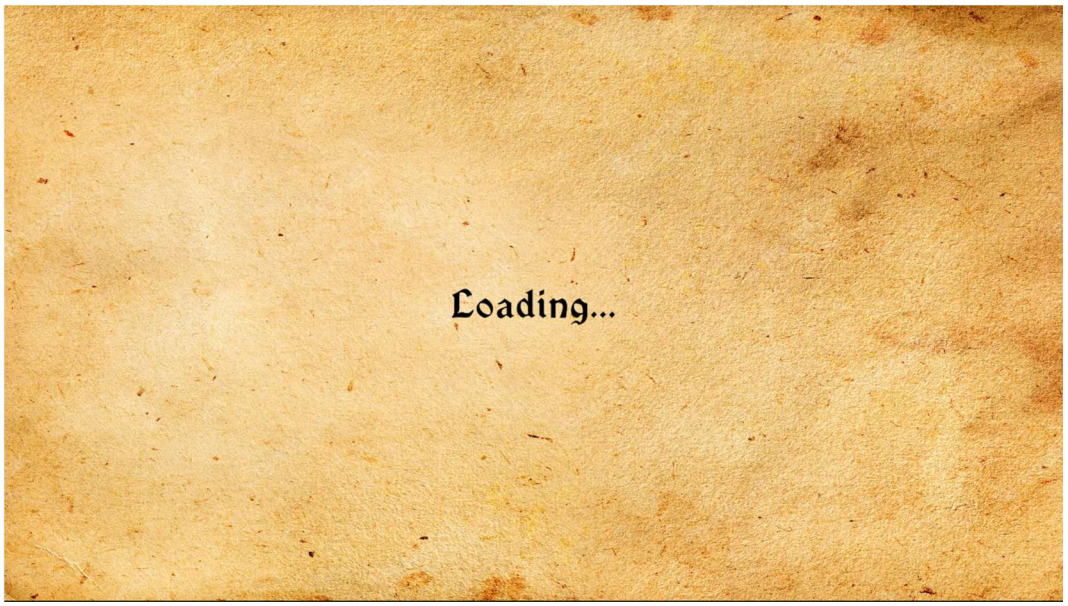


Figura 33: Pantalla de càrrega

En el cas que s'estigui iniciant una partida nova, abans es mostrarà una pantalla de tutorial (Figura 34) amb una breu explicació dels objectius.

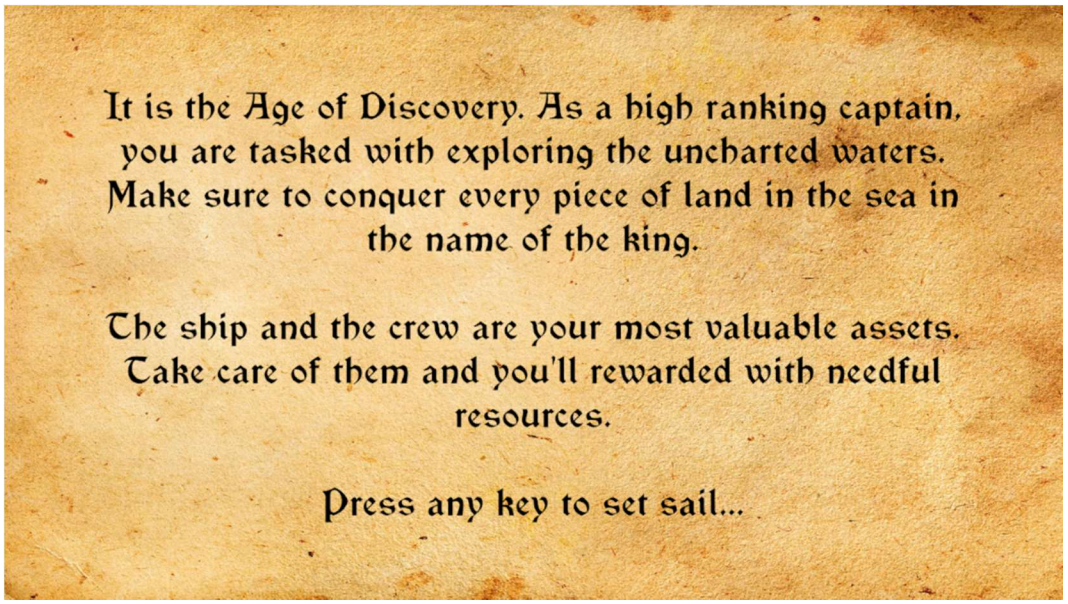


Figura 34: Pantalla de tutorial

Durant el joc es pot clicar la tecla Esc per obrir i tancar el menú de pausa (Figura 35). Aquest es presenta com un popup al mig de la pantalla amb els botons "Guardar", "Guardar i sortir" i "Sortir sense guardar". Com en tots els popups, hi ha una creueta a la cantonada superior esquerra que permet tancar-lo.

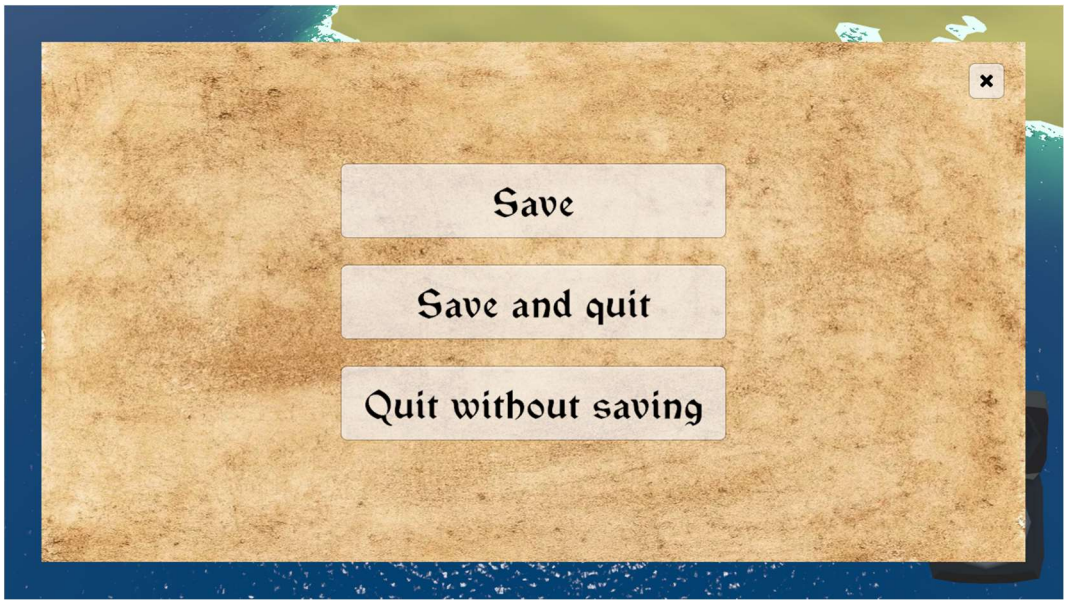


Figura 35: Menú de pausa

5.5.2. Icones

Per aconseguir una estètica coherent a través dels menús s'han obtingut totes les icones de Flaticon, un web amb milions d'imatges vectorials gratuïtes i disponibles per a ús personal i comercial. S'hi poden trobar col·leccions d'icones, el qual garanteix la sinergia entre elles. En altres casos, s'han utilitzat els mateixos filtres de color i contorn i s'han buscat els resultats amb els estils més similars. En la Figura 36, Figura 37, Figura 38, Figura 39 i Figura 40 es mostren les icones utilitzades en el projecte.

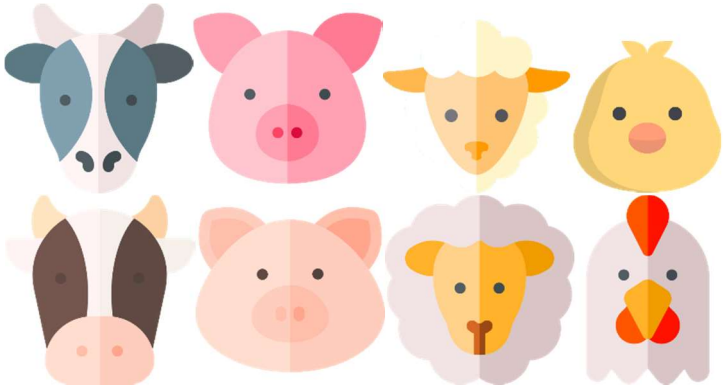


Figura 36: Icones dels animals. De dalt a baix i d'esquerra a dreta: vedell, vaca, garrí, porc, xai, ovella, pollet i gallina

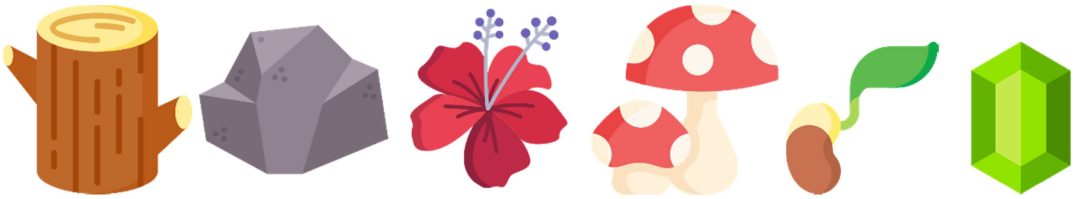


Figura 37: Icones dels materials. D'esquerra a dreta: fusta, pedra, flor, bolet, llavor d'arbre i gemma

Figura 38: Icones de les verdures: D'esquerra a dreta i de dalt a baix: ceba, pastanaga, albergínia, cogombre, enciam, patata, tomàquet, carbassó, pebrot i blat de moro

Figura 39: Icones de les carns. A dalt, d'esquerra a dreta: vedella, carn de porc, carn d'ovella, pollastre i peix. A baix, la versió cuita de la fila de dalt.

Figura 40: Icones de les construccions: D'esquerra a dreta: hort, corral, taverna, casa, mina, magatzem i vaixell

5.5.3. Botons de navegació

S'han dissenyat els menús perquè siguin minimalistes i donin al jugador la informació més important de la manera més intuïtiva possible.

Mentre el vaixell està navegant, es mostra la brúixola (Figura 41) a la cantonada inferior dreta. Per deixar clar que l'objecte es troba al vaixell, la imatge amb els punts cardinals rota d'acord amb la orientació que tingui i es balanceja alhora. Té una agulla vermella que indica el destí que hagi decidit el jugador.

Figura 41: Brúixula

Si el jugador clica la brúixola s'obrirà el mapa (Figura 42). Es mostra com si fos la vista del capità, mirant el pergamí sobre una taula de fusta. El vaixell i les illes es representen amb dibuixos, com en un mapa antic. El destí es veu com una agulla clavada a la ubicació d'interès, i es pot canviar clicant en algun altre punt del mapa.

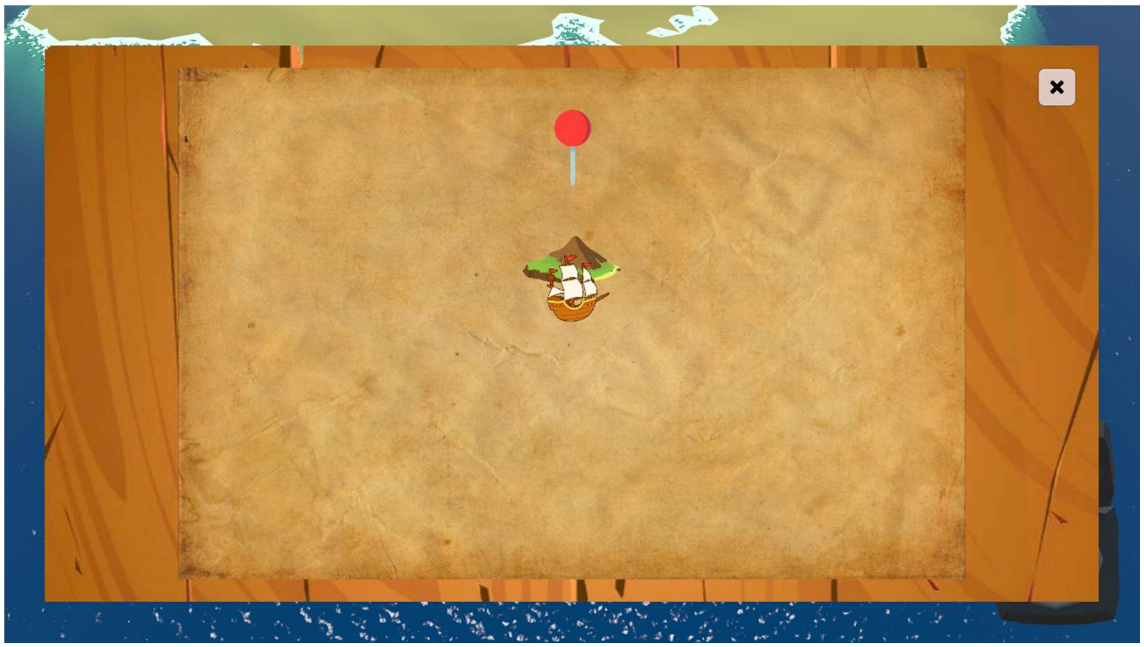


Figura 42: Mapa

Si el vaixell es troba en una zona adient es mostrarà l'opció de pescar. El botó (Figura 43) apareix a la cantonada inferior esquerra amb el text "Començar a pescar". Si es clica el text canviarà a "Deixar de pescar".

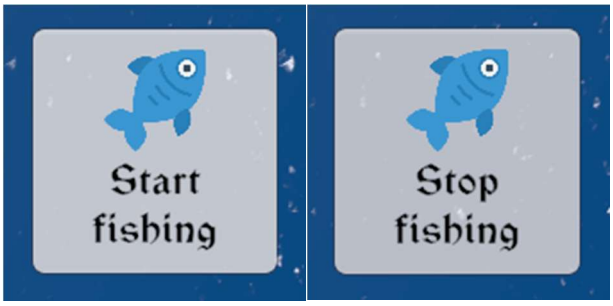


Figura 43: Botó de pescar

Si el jugador es creua amb el vaixell enemic i aquest està obert a intercanviar recursos apareixerà el botó "Comerciar" (Figura 44).

Figura 44: Botó "Comerciar"

Quan cliqui el botó s'obrirà el menú d'intercanvis (Figura 45). Es presenten dues llistes, una amb compres i l'altra amb vendes. Quan s'efectuï una transacció quedarà desactivada.

Figura 45: Menú d'intercanvis

Quan el vaixell estigui suficientment a prop d'una illa com per desembarcar, apareixerà el botó per fer-ho (Figura 46) a la part inferior central. Si l'illa ha estat descoberta, es mostrarà el seu nom.

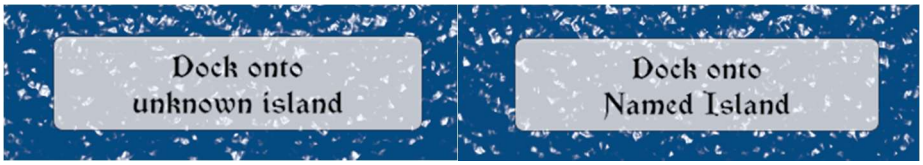


Figura 46: Botó per desembarcar. A l'esquerra, l'illa no ha estat descoberta. A la dreta, l'illa ha estat descoberta

Si desembarca el botó substituït pel d'embarcar (Figura 47). La brúixola s'amagarà.

Figura 47: Botó d'embarcar

A més, si la illa no havia estat descoberta as mostrarà el popup per anomenar-la (Figura 48). La caixa de text només acceptarà una col·lecció de paraules amb un màxim de 20 caràcters. Automàticament es capitalitza la primera lletra de cada paraula.

Figura 48: Popup per anomenar l'illa

5.5.4. Botons d'edició

Mentre el jugador estigui en una illa es mostraran les tres pestanyes d'edició de l'illa (Figura 49) a la part superior, cada una amb tres botons. Aquests comunicaran al joc quina acció vol realitzar el jugador. Quan es selecciona una pestanya es mostren els seus botons, i si es torna a seleccionar es tornen a amagar.

Figura 49: Pestanyes d'edició de l'illa

La pestanya "servei" conté les accions de construir un magatzem, una casa i una taverna. La pestanya "treball" conté les de construir un hort, un corral i una mina. Finalment, la pestanya "elements" conté les de plantar arbres i les de marcar i desmarcar elements com a pendents de treure. Quasi totes les accions requereixen recursos, i és possible que no es puguin realitzar perquè en manca algun. Si fos el cas el botó queda desactivat i s'assenyala el recurs que falta amb text vermell. En la Figura 50 es mostra cada pestanya oberta.

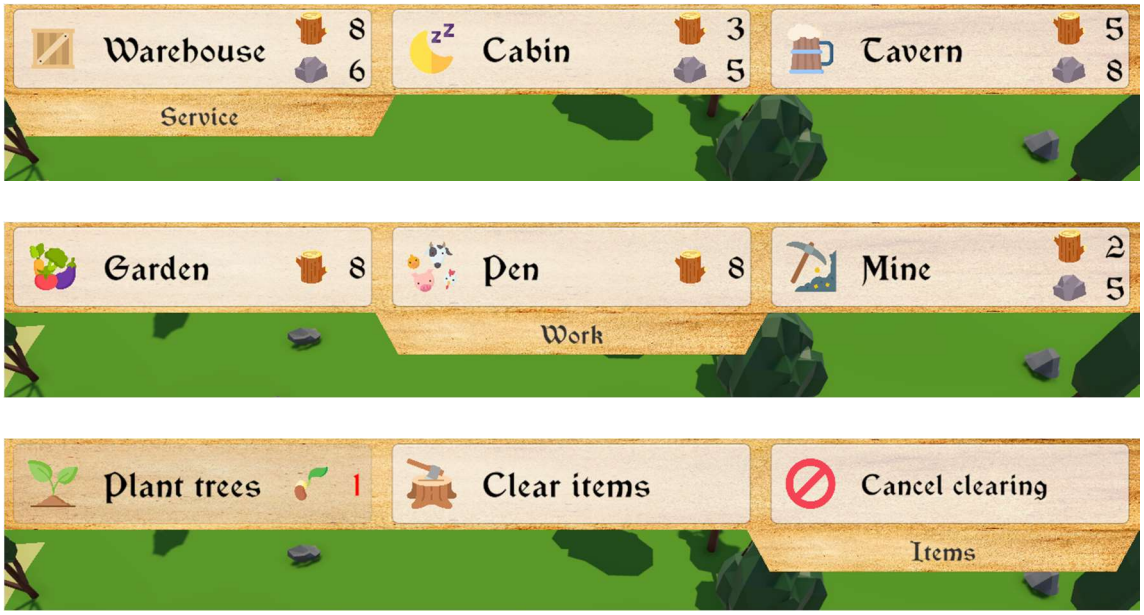


Figura 50: Pestanyes d'edició de l'illa, obertes. De dalt a baix: la pestanya "servei", la pestanya "treball" i la pestanya "elements"

Quan es seleccioni una construcció es mostraran els seus detalls (Figura 51) a la part inferior de la pantalla. Aquests inclouran la icona, el tipus de construcció i la quantitat de personatges que alberga. També hi ha els botons per editar la construcció, destruir-la o amagar els detalls. Per enviar personatges a dins o a fora de la construcció només cal clicar els botons amb els símbols de sumar i restar. Es pot seleccionar la següent construcció de la llista clicant els botons de les fletxes.

Figura 51: Detalls d'una construcció

Si es clica el botó d'editar la construcció apareix el menú d'edició d'aquesta. Cada tipus de construcció té un menú propi que permet modificar-ne certs aspectes únics. La posició del botó que puja el nivell de la construcció pot canviar d'ubicació.

El menú d'edició de l'hort (Figura 52) permet canviar el tipus de verdura que es cultivarà en cada cel·la. Per fer-ho cal seleccionar la verdura desitjada en el desplegable de la cantonada superior esquerra i clicar les cel·les on es vulgui cultivar.

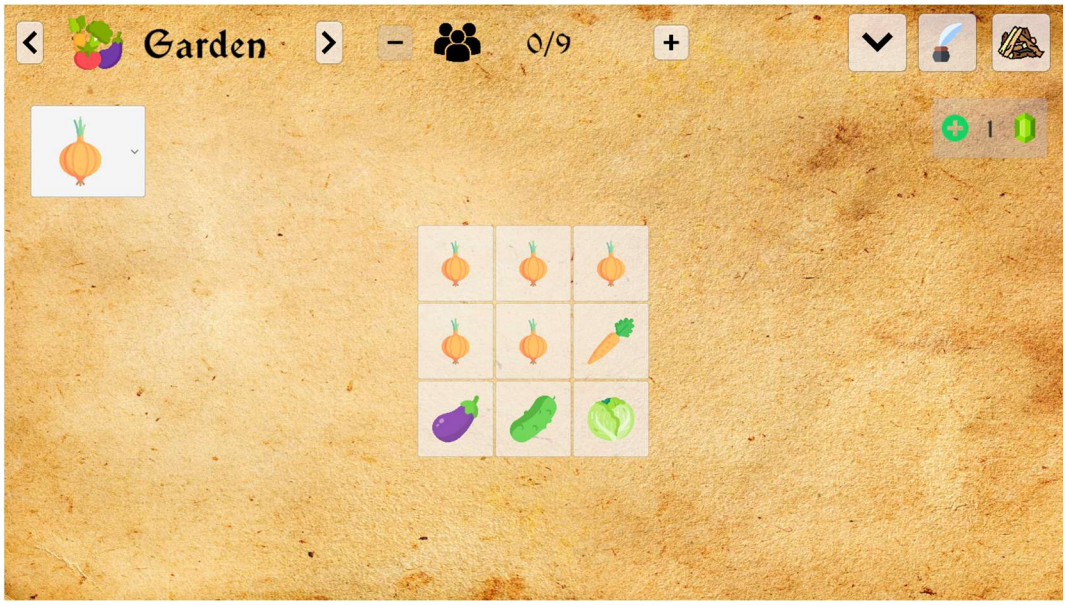


Figura 52: Menú d'edició d'un hort

El menú d'edició del corral (Figura 53) dóna l'opció d'intercanviar animals amb el vaixell a través dels botons de les fletxes. En la part superior es mostra la capacitat del vaixell i el tancat. Els animals adults tenen uns controls addicionals que permeten indicar la quantitat desitjada del tipus en qüestió. Els tipus d'animals que no estiguin en cap de les dues ubicacions es mostraran com a no disponibles.

Figura 53: Menú d'edició d'un corral

En el menú d'edició de la casa (Figura 54) es llisten tots els personatges. Es veu el cap, el nivell d'edat, el de gana i el de cansament de cada un. Estan dividits en tres llistes: "a l'illa", "en una construcció" i "al vaixell". Es pot seleccionar un personatge amb el botó de l'agulla, sempre que no es trobi dins un edifici o el vaixell. Llavors es tancarà el menú d'edició i la càmera es mourà per enfocar-lo.

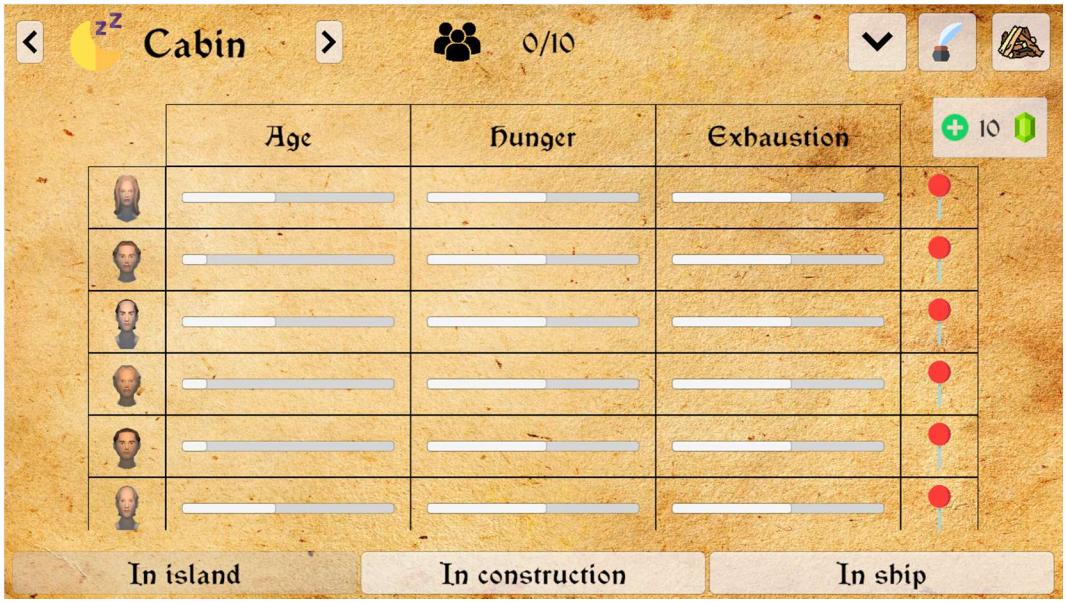


Figura 54: Menú d'edició d'una casa

El menú d'edició de la taverna (Figura 55) permet dissenyar les receptes que consumiran els personatges. Cada recepta tindrà almenys una verdura introduïda, una verdura nativa o una peça de carn, i com a màxim en tindrà un de cada. Els controls també permeten afegir receptes, reordenar-les i eliminar-les.

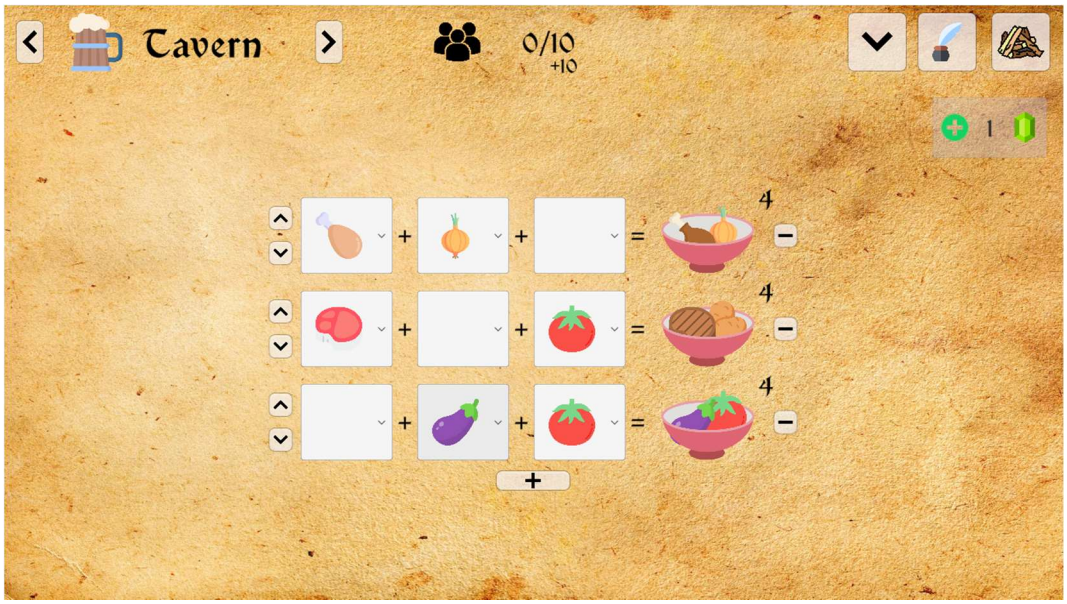


Figura 55: Menú d'edició d'una taverna

El menú d'edició de la mina (Figura 56) presenta l'opció de pujar el nivell com si s'estigués aprofundint en el terreny.

Figura 56: Menú d'edició d'una mina

En el menú d'edició del magatzem (Figura 57) es poden moure recursos entre el vaixell i l'illa, o eliminar-los per obtenir espai. De la mateixa manera que en el menú d'edició de la casa, els recursos estan dividits en tres llistes, aquest cop segons la categoria a la qual pertanyen.

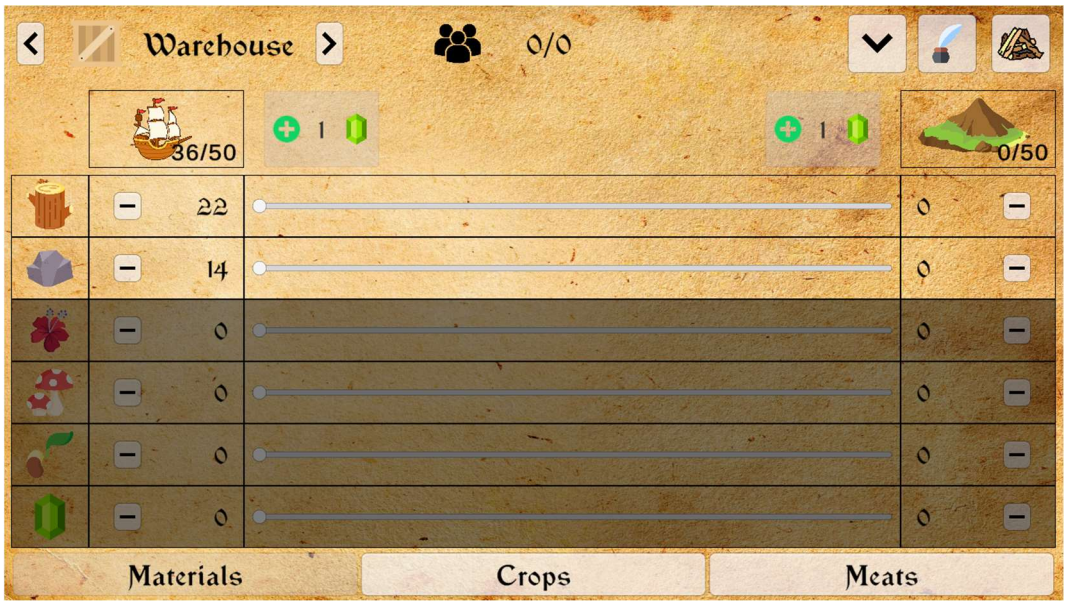


Figura 57: Menú d'edició d'un magatzem

El menú d'edició del vaixell (Figura 58) permet reparar-lo després d'haver rebut canonades durant un combat. El nivell de vida d'un vaixell es divideix en 4 seccions, separades per les parts típiques d'una embarcació: popa, proa, obra morta, obra viva. Cada secció es pinta de color verd si està al nivell màxim i de color vermell en cas contrari. Si una secció no es pot reparar, ja sigui perquè està al nivell màxim o no es tenen recursos suficients, el botó quedarà desactivat.

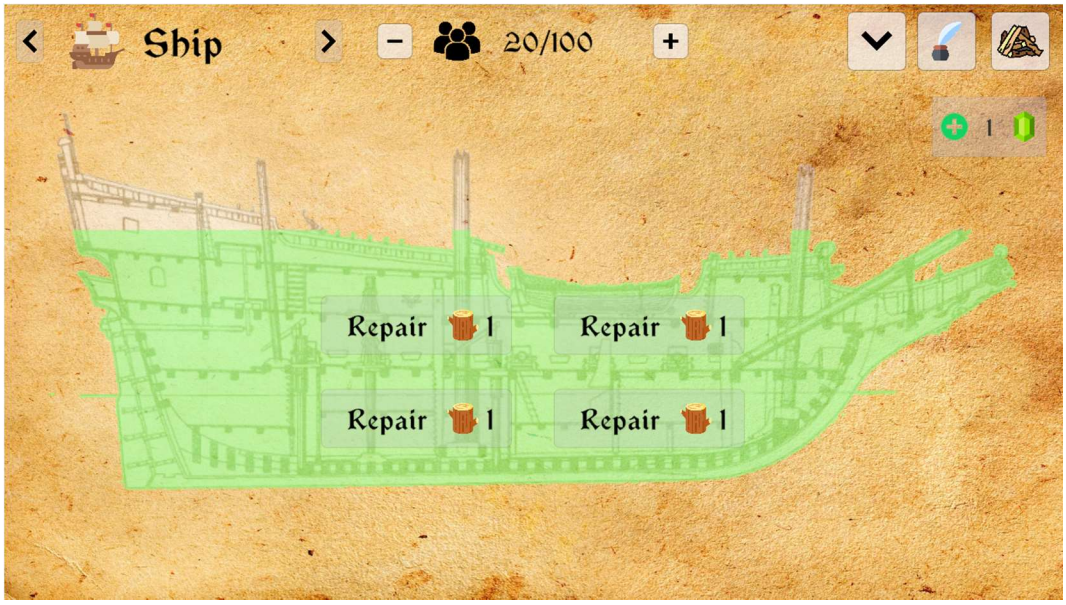


Figura 58: Menú d'edició del vaixell

Cada vegada que s'obtinguin o es gastin recursos es notificarà al costat esquerre de la pantalla amb uns indicadors (Figura 59) que només es mostraran breument. Especifiquen el tipus de recurs i la quantitat obtinguda o gastada.

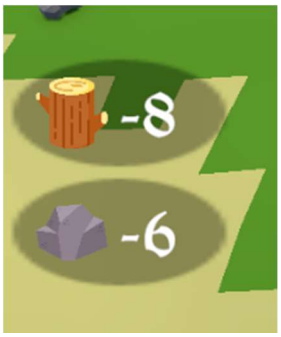


Figura 59: Indicadors dels recursos obtinguts o gastats

Quan es selecciona un personatge es mostren els seus detalls (Figura 60) a la part inferior de la pantalla. Amb el botó de la fletxa s'amaguen els detalls.

Figura 60: Detalls del personatge

Si un personatge té gana o està cansat li apareixerà una bombolla (Figura 61) a sobre el cap indicant que vol anar a una taverna o una casa per satisfer les seves necessitats.

Figura 61: Un personatge amb la bombolla que indica que té gana i està cansat

Quan un personatge reculli el brot d'una verdura nativa que encara no hagués estat descoberta, apareixerà un popup (Figura 62) indicant el nou tipus de verdura disponible.

Figura 62: Popup de verdura descoberta

5.6. Programari

5.6.1. Unity

Unity és una eina multiplataforma de creació de videojocs i experiències 2D i 3D. Té una increïble flexibilitat, ja que admet molts tipus de fitxers de contingut. El motor gràfic utilitza OpenGL i ShaderLab per a la il·luminació i l'ombrejat, i permet escriure shaders propis.

Unity té C# com a llenguatge de programació principal. És un llenguatge de propòsit general i multiparadigma desenvolupat per Microsoft. La seva sintaxi bàsica deriva de C/C++ i utilitza el model d'objectes de la plataforma .NET, el qual és similar al de Java però inclou millores derivades d'altres llenguatges. C# va ser dissenyat per a combinar el control a nivell baix de llenguatges com C i la velocitat de programació de llenguatges com Visual Basic.

S'ha escollit Unity com a motor de videojocs per l'experiència prèvia que es té amb aquest en particular. A més compta amb una extensa documentació i una comunitat molt activa, el qual garanteix que no hi haurà grans obstacles al llarg del desenvolupament. En la Figura 63 es mostra una captura del programa.

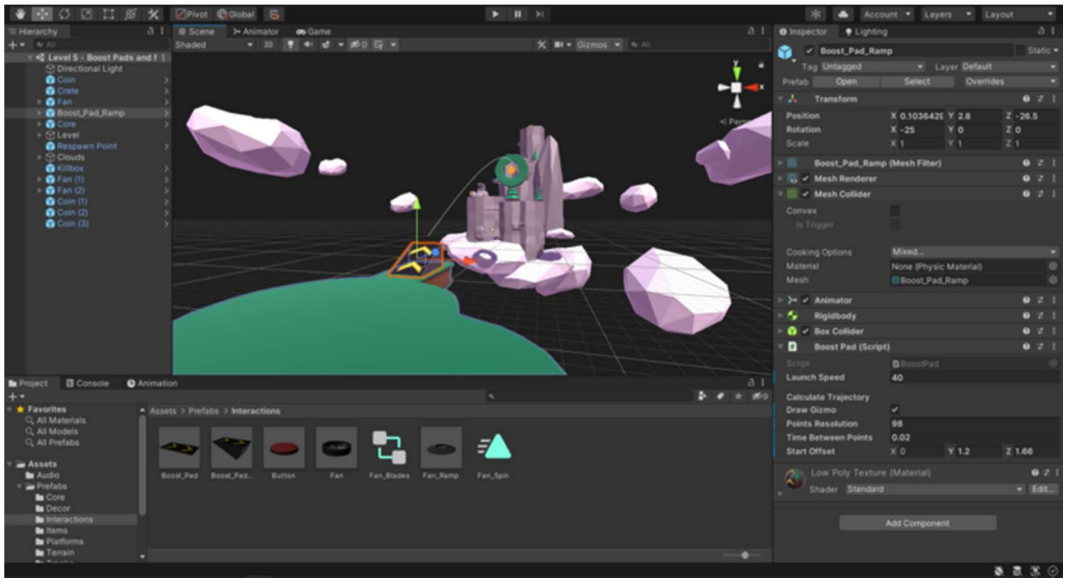


Figura 63: Unity

5.6.2. Visual Studio

Visual Studio és un entorn de desenvolupament integrat (IDE) desenvolupat per Microsoft i disponible per a Windows i macOS. És compatible amb múltiples llenguatges de programació, com ara C ++, C #, Visual Basic.NET, F #, Java, Python, Ruby i PHP, a l'igual que entorns de desenvolupament web, com ASP.NET MVC, Django, etc.

Visual Studio inclou eines potents com IntelliSense, que assisteix al programador suggerint codi. A més permet refactoritzar i debugar de forma nativa, entre altres característiques. Admet

plugins que expandeixen la seva funcionalitat en tots els nivells, incloent el suport per sistemes de control de versions com Git, i afegir eines d'edició i disseny visual.

S'ha escollit Visual Studio com a entorn de desenvolupament per la seva fantàstica eina de debugat per Unity, que permet aturar el joc en línies de codi específiques, obtenir el valor de les variables, observar la jerarquia de crides a mètodes, buscar totes les referències a través del projecte, etc. En la Figura 64 es mostra una captura del programa.

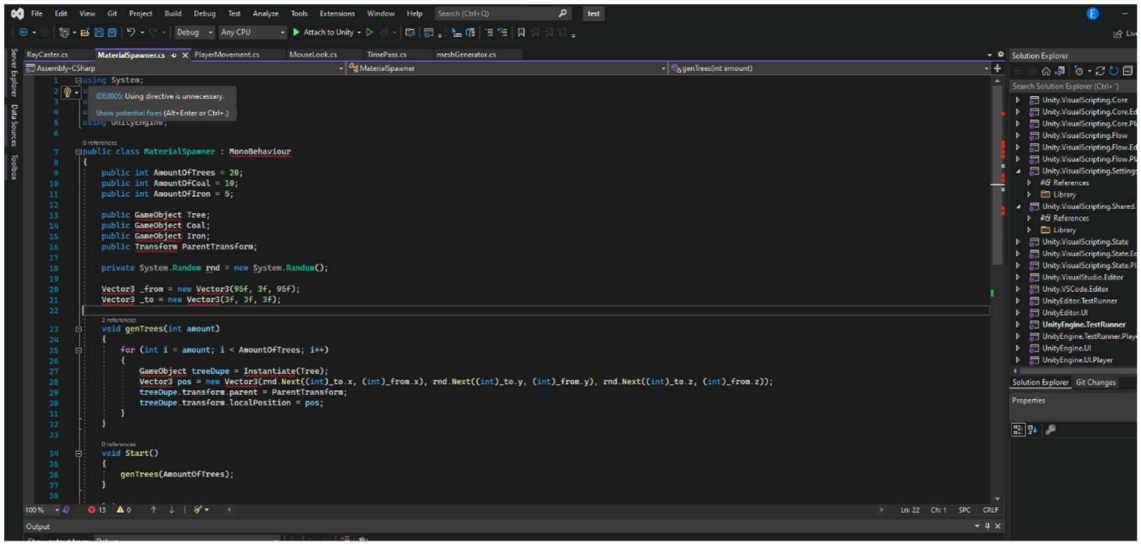


Figura 64: Visual Studio

5.6.3. GIMP

GIMP (GNU Image Manipulation Program) és un programa per al tractament d'imatges, creat per voluntaris i distribuït sota la llicència GPL. Està construït amb les llibreries GTK les quals es van crear pensant en aquest programa. Forma part del Projecte GNU.

GIMP serveix per processar gràfics i fotografies digitals. Els usos típics inclouen la creació de gràfics i logotips, el canvi de mida i retallat de fotografies, el canvi de colors, la combinació d'imatges usant capes, l'eliminació d'elements no desitjats de les imatges i la conversió entre diferents formats d'imatges. També es pot utilitzar el GIMP per crear imatges animades senzilles.

S'ha escollit GIMP com a programa de tractament d'imatges per la gran quantitat de funcionalitats que ofereix tot i ser programari gratuït. En la Figura 65 es mostra una captura del programa.

Figura 65: GIMP

5.6.4. GitHub Desktop

Git és un sistema de control de versions que ajuda a gestionar arxius de codi, entre altres, i fer-ne un seguiment. GitHub és una plataforma d'allotjament basada en núvol que permet als desenvolupadors gestionar els seus repositoris Git.

GitHub Desktop és una aplicació de codi obert que permet interactuar amb GitHub mitjançant una interfície gràfica d'usuari (GUI) en lloc de línies de comandes o un navegador web. Permet als desenvolupadors de llançar ordres com ara la creació de repositoris, la consulta de l'estat d'una branca i les sol·licituds de push i pull a través de clics. Aquesta comoditat afegeix un element addicional de flexibilitat per treballar amb Git i col·laborar amb altres desenvolupadors que no tinguin prou coneixement sobre el sistema de control de versions.

S'ha escollit GitHub pel control de versions per la facilitat d'ús de GitHub Desktop. En la Figura 66 es mostra una captura del programa.

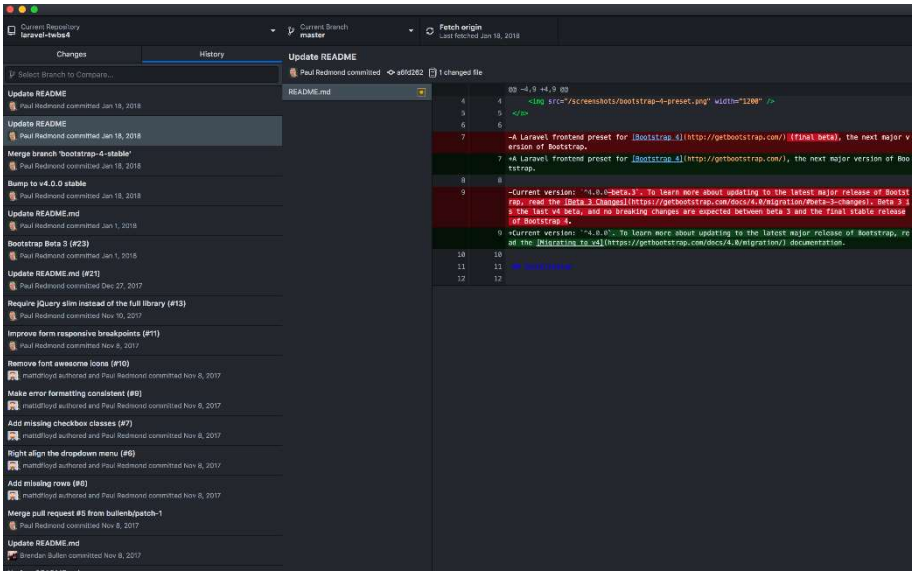


Figura 66: GitHub

6. Desenvolupament

Al llarg d'aquest punt es presenta en detall la implementació del joc.

6.1. Shading de l'aigua

Per a l'aigua del mar, es volia trobar una solució no costosa (ja que el mapa seria extens), però alhora atractiva. Es van provar algunes opcions, i les següents van resultar ser les més interessants:

6.1.1. Crear el mesh a través de codi

A l'inici de les proves es va veure un tutorial que explicava com crear un mesh i modificar les posicions dels vèrtexs a través de codi. El creador del projecte utilitzava un algorisme parametritzat que, sumat a la funció de soroll de Perlin, crea un moviment suau i aleatori semblant al de les onades del mar. A més, el tutorial incloïa el codi que simula la flotació d'objectes a l'aigua segons la posició, el pes i el centre de gravetat. Es va pensar que, com que és una solució a un nivell d'implementació tan baix, seria la menys costosa. Tot i així, escalant el mesh es va observar de l'enorme quantitat de recursos que consumia, així que es va abandonar aquesta opció. Els resultats d'utilitzar la solució es poden veure en la Figura 67 i la Figura 68.

Figura 67: Aigua creada a través de codi

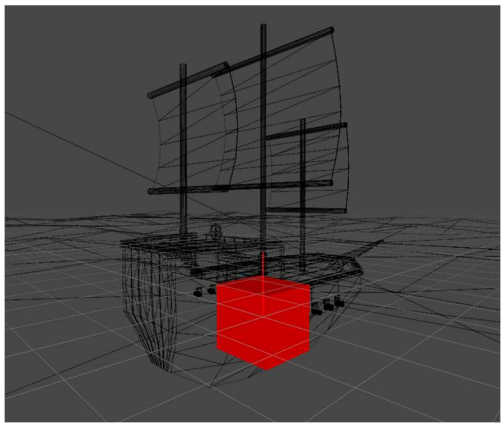


Figura 68: Aigua creada a través de codi (wireframe)

6.1.2. Crear un shader

Tot seguit es va plantejar explorar l'opció dels shaders. Va ser investigant sobre el tema que es va descobrir URP o Universal Render Pipeline, que a diferència de la Built-in RP és customitzable (scriptable). Això permet treballar amb shader graphs, una eina gràfica de creació de shaders, i en general té un millor rendiment. Per fer les proves es va haver de convertir els materials que feien servir Built-in RP a URP, que es pot fer manualment a l'inspector o amb l'eina "Convert Selected Built-in Materials to URP" (Edit>Rendering>Materials).

El shader creat aproxima càustiques, fenòmens òptics causats per la llum travessant substàncies com l'aigua. Ho fa amb l'ajuda del soroll Worley, que parteix la textura en regions basant-se en els diagrames Voronoi. La funció del soroll obté una imatge monocromàtica de degradats que, donat un color de base, recorda a les onades del mar. De la mateixa manera que en l'anterior opció, el shader genera el moviment a través de soroll de Perlin. El shader graph resultant es pot veure en la Figura 69. Els resultats obtinguts (Figura 70) eren interessants però no del tot satisfactoris, així que vaig seguir investigant.

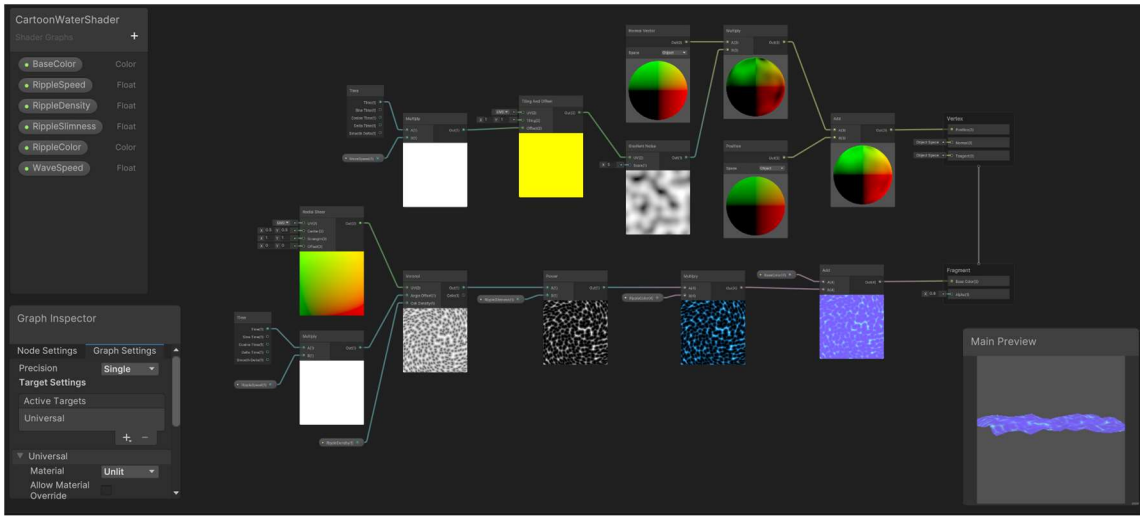


Figura 69: Shader graph de l'aigua

Figura 70: Aigua renderitzada amb shader graph

6.1.3. Usar un shader existent

Es va decidir fer servir un shader ja desenvolupat i optimitzat. Una opció era el que va ser creat per l'equip Unity per demostrar la potència d'URP. Aquest, igual que la primera opció que es va considerar, modifica el wireframe i fa servir físiques per balancejar els objectes flotants, però és poc estable. Finalment vam trobar un paquet a la Unity Asset Store creat per Alexander Ameye que va semblar ideal per a les nostres necessitats. És fàcil d'utilitzar, poc costós i alhora atractiu. Es mostren els resultats de les dues solucions en la Figura 71 i la Figura 72, respectivament.

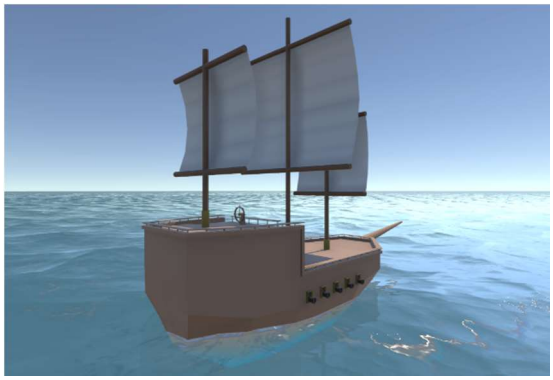


Figura 71: Aigua renderitzada amb el shader creat per l'equip de Unity

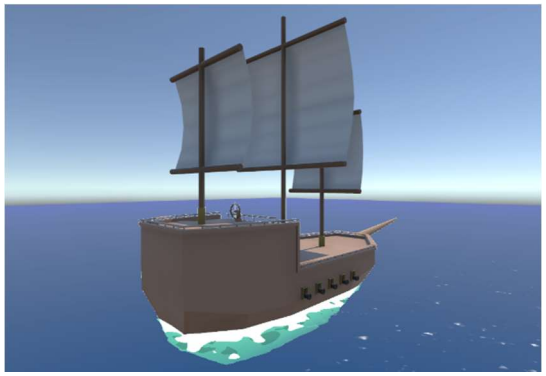


Figura 72: Aigua renderitzada amb el shader creat per Alexander Ameye

6.2. Generació de les illes

Sebastian Lague, divulgador en el sector de la programació, ha creat una sèrie de vídeos molt exhaustiva que no només dona exemples de codi sinó que a més explica els conceptes matemàtics que hi havia al darrere. A continuació, i basat en aquest tutorial, s'exposa pas per pas el procés per generar illes de manera procedural a Unity.

6.2.1. Mapa de soroll

A Unity es pot obtenir un mapa de soroll amb els mètodes incorporats al motor. La funció Mathf.PerlinNoise retorna un valor entre 0 i 1 basant-se en unes coordenades x i y passades per paràmetre. Si es crea una array bidimensional, s'itera per totes les posicions i es crida aquella funció per cada posició s'obtindrà un mapa de valors que segueixen una aleatorietat coherent. El mètode GenerateNoiseMap de la classe Noise executa aquesta lògica, tal i com es veu a la Figura 73.

```
public static class Noise {

    public static float[,] GenerateNoiseMap(int mapWidth, int mapHeight, float scale) {
        float[,] noiseMap = new float[mapWidth,mapHeight];

        if (scale <= 0) {
            scale = 0.0001f;
        }

        for (int y = 0; y < mapHeight; y++) {
            for (int x = 0; x < mapWidth; x++) {
                float sampleX = x / scale;
                float sampleY = y / scale;

                float perlinValue = Mathf.PerlinNoise (sampleX, sampleY);
                noiseMap [x, y] = perlinValue;
            }
        }

        return noiseMap;
    }

}
```

Figura 73: Classe Noise

Aquest mapa es converteix llavors en un mapa de color, en què cada valor es tradueix en la distància entre blanc i negre. La funció DrawNoiseMap de la classe MapDisplay que converteix l'array de soroll en una textura, com es pot veure a la figura Figura 74.

```
public class MapDisplay : MonoBehaviour {

    public Renderer textureRender;

    public void DrawNoiseMap(float[,] noiseMap) {
        int width = noiseMap.GetLength (0);
        int height = noiseMap.GetLength (1);

        Texture2D texture = new Texture2D (width, height);

        Color[] colourMap = new Color[width * height];
        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                colourMap [y * width + x] = Color.Lerp (Color.black, Color.white, noiseMap [x, y]);
            }
        }
        texture.SetPixels (colourMap);
        texture.Apply ();

        textureRender.sharedMaterial.mainTexture = texture;
        textureRender.transform.localScale = new Vector3 (width, 1, height);
    }

}
```

Figura 74: Classe MapDisplay

Finalment es pot crear una textura basada en el mapa de color i aplicar-la al material d'un pla. El mètode GenerateMap de la classe MapGenerator s'encarrega de combinar els resultats, com es pot veure a la figura Figura 75.

```
public class MapGenerator : MonoBehaviour {

    public int mapWidth;
    public int mapHeight;
    public float noiseScale;

    public bool autoUpdate;

    public void GenerateMap() {
        float[,] noiseMap = Noise.GenerateNoiseMap (mapWidth, mapHeight, noiseScale);


        MapDisplay display = FindObjectOfType<MapDisplay> ();
        display.DrawNoiseMap (noiseMap);
    }

}
```

Figura 75: Classe MapGenerator

La imatge resultant serà la representació de Perlin Noise projectada en un espai 3D. En la Figura 76 i la Figura 77 es pot veure la diferència que suposa modificar l'escala del soroll.

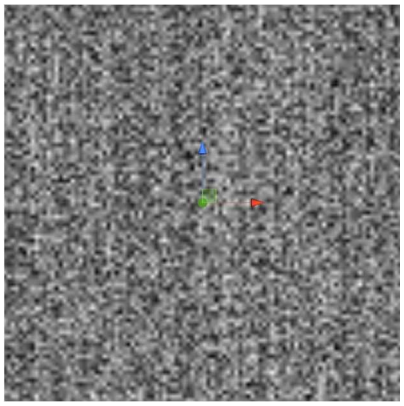


Figura 76: Soroll amb escala 0.3

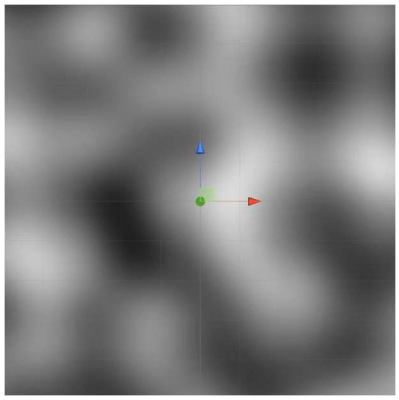


Figura 77: Soroll amb escala 27.6

Per afegir detall al mapa de soroll es combinen múltiples còpies de la mateixa funció anomenades octaves. El principi és que cada conseqüent octava tingui un major nivell de detall però un impacte més petit en el resultat. Com a exemple, en una muntanya les roques són més grans que les pedretes i per tant l'impacte serà més visible i alhora més regular. L'augment de la freqüència entre octaves és la lacunaritat, i la disminució d'amplitud entre octaves és la persistència. En la figura Figura 78 es mostra un exemple gràfic.

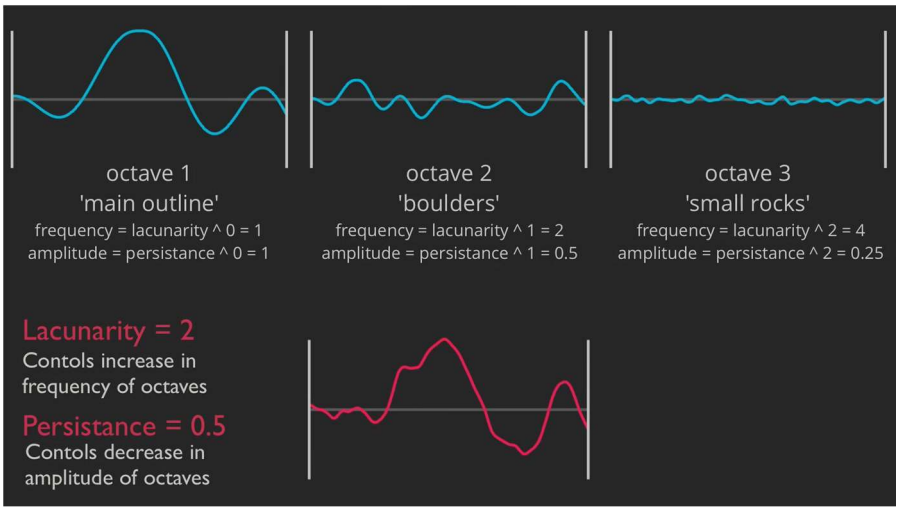


Figura 78: Esquema d'exemple de suma d'octaves

El següent pas, per tant, és afegir a la funció de creació de soroll els paràmetres d'octaves, persistència i lacunaritat. Per cada octava es tindrà en compte la variació en la freqüència i amplitud respecte a l'anterior i, després de generar els valors, es normalitzarà l'array pels valors màxims i mínims aconseguits. A la Figura 79 es veu el mètode GenerateNoiseMap, que s'ha modificat per incloure aquests conceptes en la generació del mapa de soroll.

```
public static class Noise {

    public static float[,] GenerateNoiseMap(int mapWidth, int mapHeight, float scale, int octaves, float persistance, float lacunarity) {
        float[,] noiseMap = new float[mapWidth,mapHeight];

        if (scale <= 0) {
            scale = 0.0001f;
        }

        float maxNoiseHeight = float.MinValue;
        float minNoiseHeight = float.MaxValue;

        for (int y = 0; y < mapHeight; y++) {
            for (int x = 0; x < mapWidth; x++) {

                float amplitude = 1;
                float frequency = 1;
                float noiseHeight = 0;

                for (int i = 0; i < octaves; i++) {
                    float sampleX = x / scale * frequency;
                    float sampleY = y / scale * frequency;

                    float perlinValue = Mathf.PerlinNoise (sampleX, sampleY) * 2 - 1;
                    noiseHeight += perlinValue * amplitude;

                    amplitude *= persistance;
                    frequency *= lacunarity;
                }

                if (noiseHeight > maxNoiseHeight) {
                    maxNoiseHeight = noiseHeight;
                } else if (noiseHeight < minNoiseHeight) {
                    minNoiseHeight = noiseHeight;
                }
                noiseMap [x, y] = noiseHeight;
            }
        }

        for (int y = 0; y < mapHeight; y++) {
            for (int x = 0; x < mapWidth; x++) {
                noiseMap [x, y] = Mathf.InverseLerp (minNoiseHeight, maxNoiseHeight, noiseMap [x, y]);
            }
        }

        return noiseMap;
    }
}
```

Figura 79: Classe Noise amb els paràmetres d'octaves, persistència i lacunaritat.

El soroll resultant és conegut com soroll fractal. La Figura 80 mostra soroll amb una sola octava, mentre que la Figura 81 mostra el resultat d'utilitzar 4 octaves, una persistència de 0.5 i una lacunaritat de 2.

Figura 80: Soroll amb 1 octava

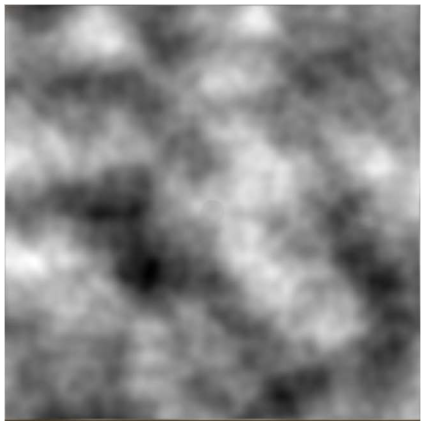


Figura 81: Soroll amb 4 octaves (fractal)

A continuació s'introdueixen els conceptes de RNG, seeds i offsets. El Random Number Generator és un objecte que retorna una seqüència de números aleatoris dins el marge especificat. Si s'especifica una seed o llavor, els números retornats seran sempre els mateixos. S'utilitzen aquests valors per definir els offsets de les octaves, és a dir, en quin punt de la funció de soroll comencen. Això serà especialment útil si es vol generar el mateix mapa de soroll més d'un cop. A la Figura 82 es poden veure els canvis.

```
public static float[,] GenerateNoiseMap(int mapWidth, int mapHeight, int seed, float scale, int octaves, float persistance, float lacunarity, Vector2 offset) {
    float[,] noiseMap = new float[mapWidth,mapHeight];

    System.Random prng = new System.Random (seed);
    Vector2[] octaveOffsets = new Vector2[octaves];
    for (int i = 0; i < octaves; i++) {
        float offsetX = prng.Next (-100000, 100000) + offset.x;
        float offsetY = prng.Next (-100000, 100000) + offset.y;
        octaveOffsets [i] = new Vector2 (offsetX, offsetY);
    }
```

Figura 82: Canvis al mètode GenerateNoiseMap per incloure seed i offset

També es modifica el codi per tal que el soroll es generi a partir del centre de la imatge enlloc de la cantonada, com es pot veure en la Figura 83.

```
float halfWidth = mapWidth / 2f;
float halfHeight = mapHeight / 2f;


for (int y = 0; y < mapHeight; y++) {
    for (int x = 0; x < mapWidth; x++) {

        float amplitude = 1;
        float frequency = 1;
        float noiseHeight = 0;

        for (int i = 0; i < octaves; i++) {
            float sampleX = (x-halfWidth) / scale * frequency + octaveOffsets[i].x;
            float sampleY = (y-halfHeight) / scale * frequency + octaveOffsets[i].y;
```

Figura 83: Canvis al mètode GenerateNoiseMap per tal que es generi el soroll a partir del centre

6.2.2. Regions

Seguidament s'incorporen colors per representar les diferents regions de terreny (sorra, gespa, roques...). Es defineix el tipus de variable TerrainType i es crea una array pública que s'omplirà a l'inspector amb les regions que es vulguin veure. Abans de renderitzar la imatge s'itera per cada coordenada i es decideix la regió a la que correspon per la seva alçada (valor de 0 a 1). En la Figura 84 es veuen els canvis fets a la classe MapGenerator.

```
[System.Serializable]
public struct TerrainType {
    public string name;
    public float height;
    public Color colour;
}
```

```
public TerrainType[] regions;
```

```
public void GenerateMap() {
    float[,] noiseMap = Noise.GenerateNoiseMap (mapWidth, mapHeight, seed, noiseScale, octaves, persistance, lacunarity, offset);

    Color[] colourMap = new Color[mapWidth * mapHeight];
    for (int y = 0; y < mapHeight; y++) {
        for (int x = 0; x < mapWidth; x++) {
            float currentHeight = noiseMap [x, y];
            for (int i = 0; i < regions.Length; i++) {
                if (currentHeight <= regions [i].height) {
                    colourMap [y * mapWidth + x] = regions [i].colour;
                    break;
                }
            }
        }
    }
```

Figura 84: Canvis a la classe MapGenerator per poder renderitzar colors

Les regions definides a l'inspector són les que es mostren en la Figura 85.

Figura 85: Regions definides

La classe TextureGenerator, vista a la Figura 86, crea la textura amb el mapa de color generat i es visualitzaran les regions amb els colors corresponents.

```
public static class TextureGenerator {

    5 referencias
    public static Texture2D TextureFromColourMap(Color[] colourMap, int width, int height) {
        Texture2D texture = new Texture2D (width, height);
        texture.filterMode = FilterMode.Point;
        texture.wrapMode = TextureWrapMode.Clamp;
        texture.SetPixels (colourMap);
        texture.Apply ();
        return texture;
    }
```

Figura 86: Classe TextureGenerator, que crea les noves textures

Entre la Figura 87 i la Figura 88 es mostra la diferència entre dues versions del mateix soroll (generat a partir dels mateixos paràmetres). La única diferència és que la segona ha estat pintada seguint la llista de regions definida. Es pot apreciar com els punts més foscos es representen amb aigua i els més clars amb neu.

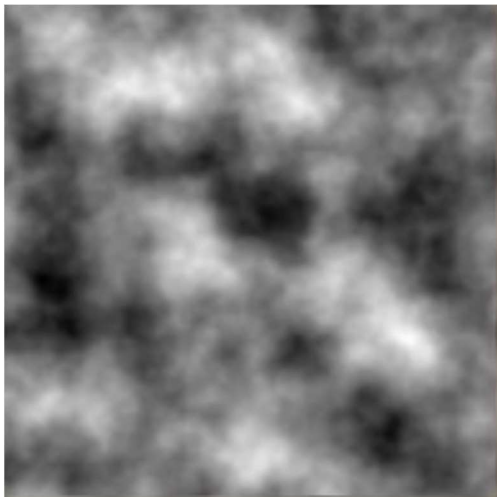


Figura 87: Mapa de soroll

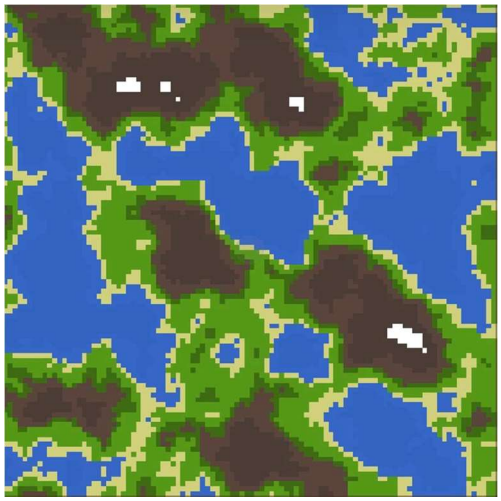


Figura 88: Mapa de soroll amb color

6.2.3. Mesh

Per aconseguir relleu a partir del mapa de soroll cal poder elevar cada coordenada a l'alçada corresponent. Per tant, s'ha de generar un mesh de les mateixes dimensions que el mapa i amb tants vèrtexs com coordenades. Els vèrtexs es guardaran en una array de vectors tridimensionals, que indicaran els valors x, y i z de cada coordenada. Els triangles es guardaran en una array d'enters, en què cada grup de 3 enters consecutius és els índexs dels vèrtexs d'un triangle. En un mesh rectangular com el d'un pla, el nombre de vèrtexs serà el producte de l'amplada i la llargada. Per saber el nombre de triangles cal pensar en la quantitat de quadrats que es podrien formar (un per cada fila i columna excepte la última) i en que cada quadrat tindrà dos triangles amb tres vèrtexs cadascun. És a dir, la mida de l'array de triangles serà el producte de l'amplada menys un i la llargada menys un multiplicat pels 6 vèrtexs dels triangles del quadrat. A la Figura 89 es pot veure la representació dels càlculs.

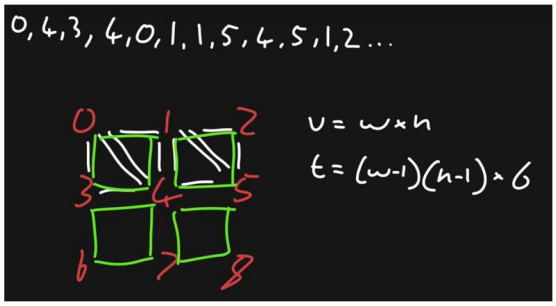


Figura 89: Dibuix amb l'explicació dels càlculs per saber la mida de les arrays de vèrtexs i triangles

També serà necessari guardar una array de UVs, que seran vectors bidimensionals amb els percentatges de x i y de la textura en els que es troba cada vèrtex. Finalment, per donar alçada a cada vèrtexs es multiplicarà el valor de la coordenada en el mapa de soroll per un factor que s'indicarà a l'inspector. Totes aquestes arrays es guardaran en un objecte de la classe MeshData, que es pot veure en la Figura 90.

```
public class MeshData {
    public Vector3[] vertices;
    public int[] triangles;
    public Vector2[] uvs;

    int triangleIndex;

    public MeshData(int meshWidth, int meshHeight) {
        vertices = new Vector3[meshWidth * meshHeight];
        uvs = new Vector2[meshWidth * meshHeight];
        triangles = new int[(meshWidth-1)*(meshHeight-1)*6];
    }

    public void AddTriangle(int a, int b, int c) {
        triangles [triangleIndex] = a;
        triangles [triangleIndex + 1] = b;
        triangles [triangleIndex + 2] = c;
        triangleIndex += 3;
    }

    public Mesh CreateMesh() {
        Mesh mesh = new Mesh ();
        mesh.vertices = vertices;
        mesh.triangles = triangles;
        mesh.uv = uvs;
        mesh.RecalculateNormals ();
        return mesh;
    }

}
```

Figura 90: Classe MeshData, que guarda les arrays de vèrtexs, triangles i UVs i crea el mesh

El mètode GenerateTerrainMesh ha estat modificat perquè ompli les variables d'un objecte MeshData, tal i com es veu a la Figura 91.

```
public static MeshData GenerateTerrainMesh(float[,] heightMap, float heightMultiplier) {
    int width = heightMap.GetLength (0);
    int height = heightMap.GetLength (1);
    float topLeftX = (width - 1) / -2f;
    float topLeftZ = (height - 1) / 2f;

    MeshData meshData = new MeshData (width, height);
    int vertexIndex = 0;

    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {

            meshData.vertices [vertexIndex] = new Vector3 (topLeftX + x, heightMap [x, y] * heightMultiplier, topLeftZ - y);
            meshData.uvs [vertexIndex] = new Vector2 (x / (float)width, y / (float)height);

            if (x < width - 1 && y < height - 1) {
                meshData.AddTriangle (vertexIndex, vertexIndex + width + 1, vertexIndex + width);
                meshData.AddTriangle (vertexIndex + width + 1, vertexIndex, vertexIndex + 1);
            }

            vertexIndex++;
        }
    }

    return meshData;

}
```

Figura 91: Classe MeshGenerator, que omple les variables d'un objecte MeshData

El problema d'utilitzar un únic factor multiplicador d'alçada per totes les zones del mapa és que pot ser interessant tenir zones més planes, com la costa, i més irregulars, com les muntanyes. Per arreglar-ho s'introdueix una variable de tipus AnimationCurve que permetrà ajustar el factor de multiplicació per cada alçada. En la Figura 92 es veu la modificació realitzada al mètode GenerateTerrainMesh.

```
public static MeshData GenerateTerrainMesh(float[,] heightMap, float heightMultiplier, AnimationCurve heightCurve) {
    int width = heightMap.GetLength (0);
    int height = heightMap.GetLength (1);
    float topLeftX = (width - 1) / -2f;
    float topLeftZ = (height - 1) / 2f;

    MeshData meshData = new MeshData (width, height);
    int vertexIndex = 0;

    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {

            meshData.vertices [vertexIndex] = new Vector3 (topLeftX + x, heightCurve.Evaluate(heightMap [x, y]) * heightMultiplier, topLeftZ - y);
```

Figura 92: Canvis en la classe MeshGenerator, amb el paràmetre heightCurve

En l'inspector es dibuixa el valor de heightCurve amb l'editor que es mostra a la Figura 93.

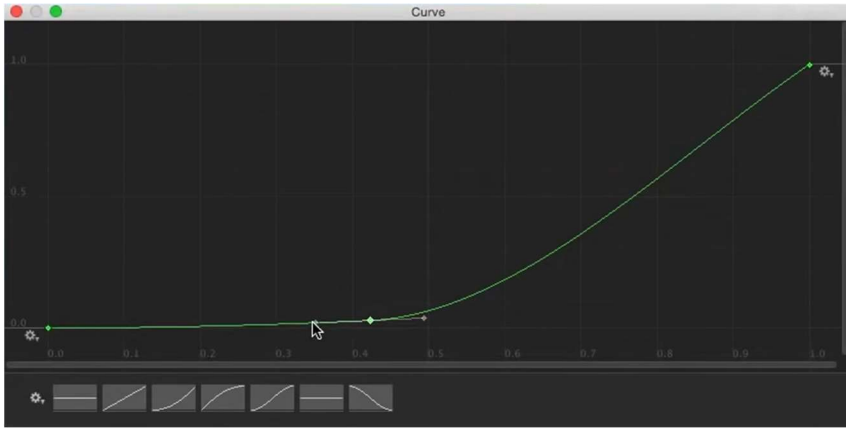


Figura 93: Finestra d'edició de la variable heightCurve

Entre la Figura 94 i la Figura 95 es pot apreciar la diferència entre el mesh generat sense heightCurve i el mesh que inclou la variable, respectivament.

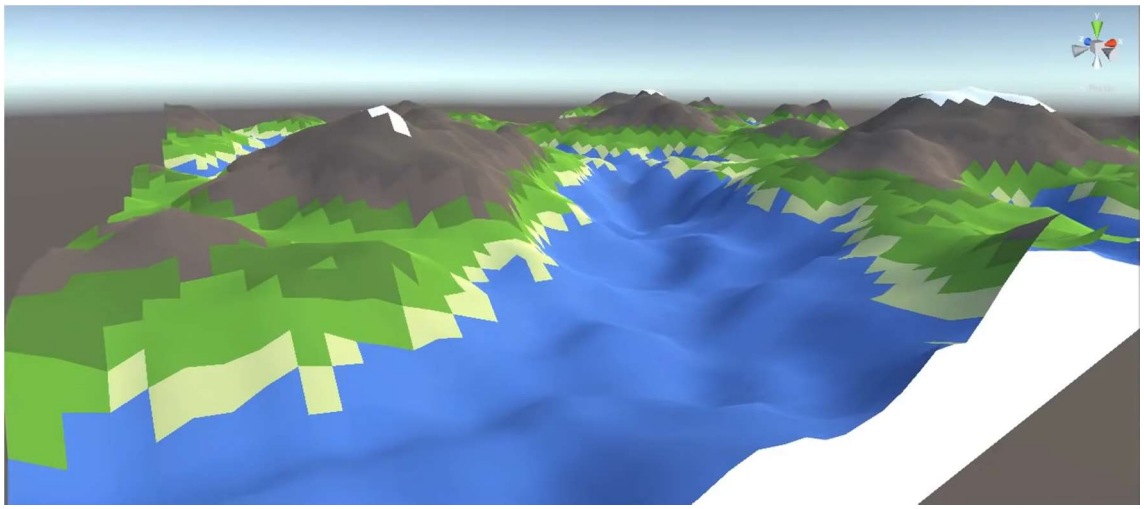


Figura 94: Mesh generat sense la variable heightCurve

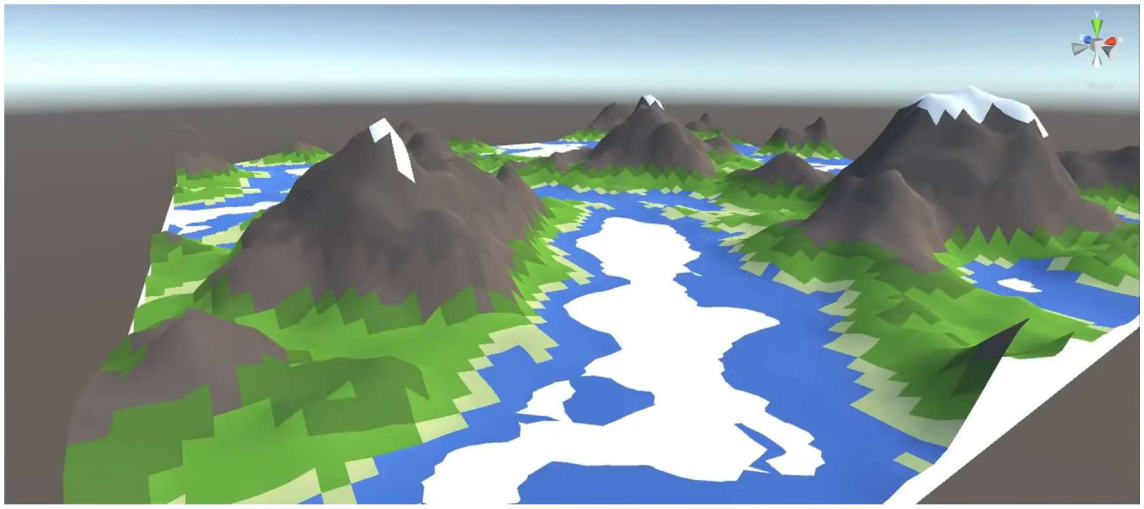


Figura 95: Mesh generat amb la variable heightCurve

Una cosa que s'ha de tenir en compte a l'hora de crear un mesh a través de codi és que Unity té un límit de 65.025 vèrtexs per mesh, és a dir, un mesh quadrat de 255x255 vèrtexs. Això no serà un problema per aquest projecte, ja que les illes tindran una mida molt més petita.

6.2.4. Falloff

Un cop ja es genera un tros de terreny, cal rebaixar les vores del mesh per aconseguir la forma d'una illa. Per aconseguir-ho s'ha de crear un "falloff map", és a dir, una imatge de les mateixes dimensions que el mapa de soroll que, combinant els valors, les alçades que es trobin a certa distància del centre siguin 0 i la pendent sigui gradual. Aquesta imatge es genera fent un bucle per a cada coordenada i calculant un valor entre 0 i 1 segons la distància màxima entre el número de fila o columna i el centre. En la Figura 96es veu el mètode GenerateFalloffMap de la classe FalloffGenerator, que crea el mapa en qüestió.

```
public static class FalloffGenerator {

    public static float[,] GenerateFalloffMap(int size) {
        float[,] map = new float[size,size];

        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                float x = i / (float)size * 2 - 1;
                float y = j / (float)size * 2 - 1;

                float value = Mathf.Max (Mathf.Abs (x), Mathf.Abs (y));
                map [i, j] = value;
            }
        }

        return map;
    }
}
```

Figura 96: Classe FalloffGenerator, que genera el falloff map

A la Figura 97 i la Figura 98 es mostra la representació gràfica del mapa generat i el resultat d'aplicar-lo al mapa de soroll de l'illa.

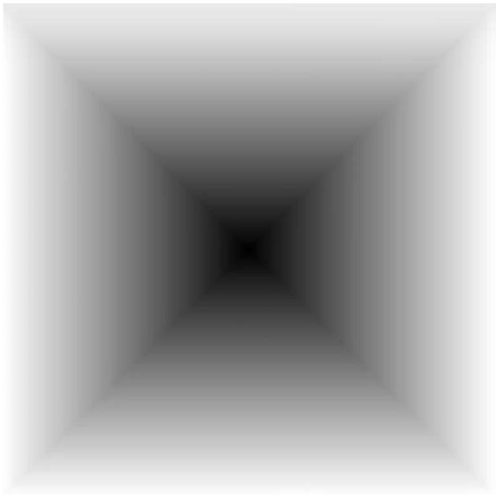


Figura 97: Falloff map

Figura 98: Falloff map aplicat al mesh

Llavors convé aplicar una equació matemàtica que ajusti el degradat del falloff map. Això permetrà regular la distància a partir de la qual es produeix la pendent i amb quina suavitat. L'equació té dues incògnites, a i b, que dicten l'angle i el punt d'inici de la pendent, respectivament. En la Figura 99es veu l'equació representada amb l'ajuda de Desmos, un programari de visualització d'equacions matemàtiques.

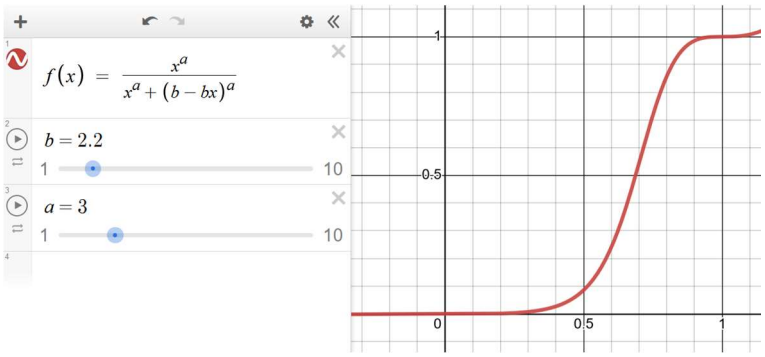


Figura 99: Equació representada amb Desmos

A la classe FalloffGenerator s'afegeix la funció Evaluate, que altera els valors del falloff aplicant l'equació. En la Figura 100 es pot veure els canvis en el mètode GenerateFalloffMap.

```
public static class FalloffGenerator {

    public static float[,] GenerateFalloffMap(int size) {
        float[,] map = new float[size,size];

        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                float x = i / (float)size * 2 - 1;
                float y = j / (float)size * 2 - 1;

                float value = Mathf.Max (Mathf.Abs (x), Mathf.Abs (y));
                map [i, j] = Evaluate(value);
            }
        }

        return map;
    }

    static float Evaluate(float value) {
        float a = 3;
        float b = 2.2f;

        return Mathf.Pow (value, a) / (Mathf.Pow (value, a) + Mathf.Pow (b - b * value, a));
    }
}
```

Figura 100: Classe FalloffGenerator amb l'equació

A la Figura 101 i la Figura 102 es repeteix la comparació del mapa de falloff i el mesh, aquest cop amb l'equació.

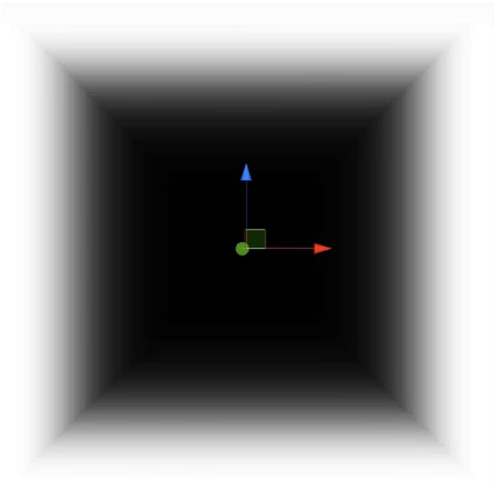


Figura 101: Falloff map amb l'equació

Figura 102: Falloff map amb l'equació aplicat al mesh

6.2.5. NavMesh

Per tal que els personatges puguin navegar per l'illa és essencial establir una NavMesh. És una propietat de l'escena que defineix quines són les àrees per les quals poden caminar. Es pot calcular en l'editor clicant el botó "Bake" de la pestanya de navegació o a través de codi cridant el mètode "BuildNavMesh" de la classe "NavMeshSurface". Aquesta pertany al paquet experimental "NavMeshComponents", que es pot incloure al projecte amb l'enllaç de GitHub.

El problema amb el mètode "BuildNavMesh" és que és síncron, i en superfícies complexes com el relleu d'una illa pot arribar a tardar uns segons, durant els quals el joc es congela. A la Figura 103 es mostra la finestra de Profiler de Unity, on es veu que s'han tardat 2,4 segons a generar el NavMesh d'una illa amb el mètode "BuildNavMesh".

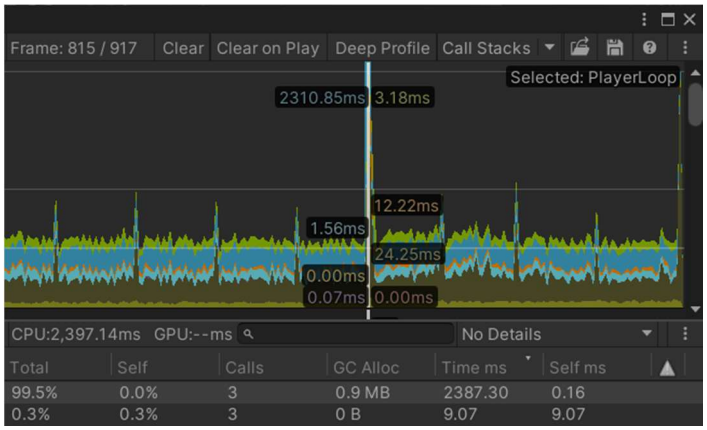


Figura 103: Finestra del Profiler de Unity

L'objectiu és aconseguir realitzar l'operació de forma asíncrona. D'aquesta manera, el joc podria fer els càlculs necessaris sense afectar massivament el rendiment. En la documentació oficial de Unity es menciona el mètode "BuildNavMeshAsync", que per desgràcia no es pot cridar a través del codi ja que està limitat a l'editor. En un fòrum que discuteix possibles solucions i en un dels comentaris hi ha l'enllaç a un article que explica en detall una manera d'eludir els controls imposats per part de l'equip de Unity.

A la Figura 104 hi ha el codi proveït per l'article i adaptat per les nostres necessitats. Primer s'obtenen els marges de l'illa i s'omple una llista amb tots els objectes a l'interior que puguin contribuir en el resultat. Després es defineix que es vol que la NavMesh serveixi per a tots els tipus d'agent definits. Seguidament es crida la funció "UpdateNavMeshDataAsync" amb tots els paràmetres declarats i s'espera a que hagi conclòs. Finalment s'afegeixen les dades obtingudes al total.

```
private IEnumerator BuildNavMeshAsync(IslandScript islandScript)
{
    // Get the list of all "sources" around us.  This is basically little gridded subsquares of our terrains.
    List<NavMeshBuildSource> buildSources = new List<NavMeshBuildSource>();

    // Set up a boundary area for the build sources collector to look at;
    Bounds islandBounds = new Bounds(islandScript.transform.position, new Vector3(mapChunkSize, meshHeightMultiplier, mapChunkSize));

    // This actually collects them
    NavMeshBuilder.CollectSources(islandBounds, -1, NavMeshCollectGeometry.PhysicsColliders, 0, new List<NavMeshBuildMarkup>(), buildSources);

    yield return null;

    // Get the settings for each of our agent "sizes" (humanoid, giant humanoid)
    NavMeshBuildSettings buildSettings = NavMesh.GetSettingsByIndex(-1);

    // Make a new mesh data object.
    NavMeshData navMeshData = new NavMeshData();

    // "Update" it from scratch.
    AsyncOperation buildOp = NavMeshBuilder.UpdateNavMeshDataAsync(navMeshData, buildSettings, buildSources, islandBounds);

    while (!buildOp.isDone) yield return null;

    NavMesh.AddNavMeshData(navMeshData);

    islandScript.navMeshSurface.navMeshData = navMeshData;

    yield return null;
}
```

Figura 104: Mètode "BuildNavMeshAsync" de la classe IslandGenerator

Per alguna raó, però, els resultats obtinguts no són els desitjats. A la Figura 105**¡Error! No se encuentra el origen de la referencia.** i la **¡Error! No se encuentra el origen de la referencia.** hi ha una comparació entre els resultats obtinguts per les diferents funcions de generació de NavMesh.

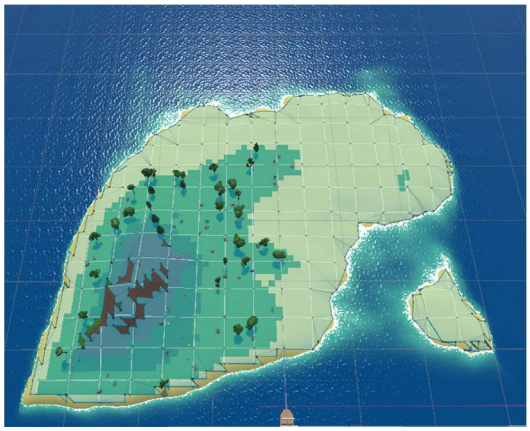


Figura 105: NavMesh generada amb BuildNavMesh

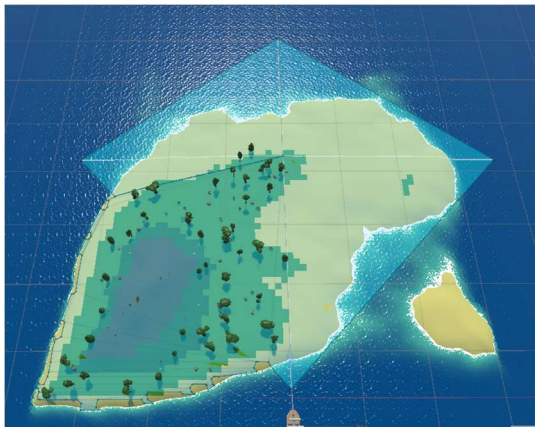


Figura 106: NavMesh generada amb BuildNavMeshAsync

Eventualment es va decidir utilitzar el mètode "UpdateNavMesh", el qual és asíncron però requereix una NavMesh ja creada. Per tant, es va reordenar l'ordre de les operacions en la creació de l'illa i es va fer que es calculés la NavMesh quasi immediatament, el qual no tarda gens ja que no hi ha geometria a tenir en compte. Després es genera el relleu i llavors es refresca la NavMesh amb el mètode asíncron, el qual tarda uns segons però no afecta al rendiment. En la Figura 107 es pot veure l'ordre d'operacions dins el mètode GenerateIsland.

```
// Part 3: Es calcula la navegació
GameObject coastObstacle = Instantiate(IslandEditor.Instance.GetCoastObstacle());
coastObstacle.transform.parent = island.transform;
coastObstacle.transform.localPosition = new Vector3(0, -3.5f, 0);

islandScript.navMeshSurface = island.AddComponent<NavMeshSurface>();
islandScript.navMeshSurface.collectObjects = CollectObjects.Children;
islandScript.navMeshSurface.BuildNavMesh();

// Part 4: Es genera el soroll a partir del seed i l'offset
MapData mapData = GenerateMapData(seed, offset);
Texture2D colorTexture = TextureGenerator.TextureFromColourMap(mapData.colourMap, mapChunkSize, mapChunkSize);
meshRenderer.material.mainTexture = colorTexture;
islandScript.regionMap = mapData.regionMap;

// Part 5: Es genera el mesh a partir del heightMap
MeshData meshData = MeshGenerator.GenerateTerrainMesh(mapData.heightMap, meshHeightMultiplier, meshHeightCurve);
Mesh mesh = meshData.CreateMesh();
meshFilter.mesh = mesh;
meshCollider.sharedMesh = mesh;
islandScript.meshData = meshData;

// Part 6: Es recalcula la navegació
islandScript.navMeshSurface.UpdateNavMesh(islandScript.navMeshSurface.navMeshData);
```

Figura 107: Fragment del mètode GenerateIsland de IslandGenerator

La nova implementació resulta en una experiència de joc molt més fluida. A la Figura 108 es veu la finestra del Profiler, però aquest cop es mesura el rendiment mentre es genera el NavMesh d'una illa amb el mètode "UpdateNavMesh". S'arriba a una frenada màxima de 82 milisegons.

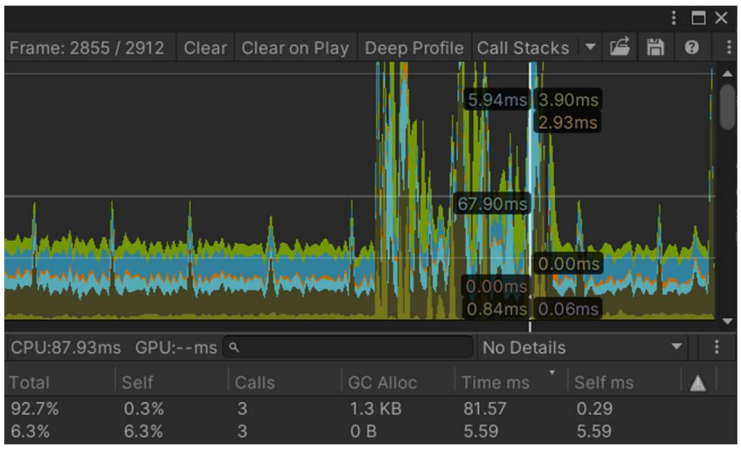


Figura 108: Finestra del Profiler de Unity

Inicialment es comptava amb un sistema de càlcul de distància entre el vaixell i l'illa més propera basat en colliders, però es va descartar per qüestions de rendiment. Es pot trobar una explicació a l'Annex 2: Generació de colliders convexos.

6.3. Edició de les illes

IslandCellScript gestiona l'edició dels elements de l'illa amb un sistema propi de cel·les. S'obté la posició del ratolí a sobre del terreny a través de la tècnica de raycasting i s'utilitza la coordenada dins l'illa per fer canvis en l'estat de la selecció.

Per detectar quan l'usuari ha començat a fer clic, ha acabat de fer clic o ha mogut el ratolí s'utilitzen els mètodes OnPointerDown, OnPointerUp i OnPointerMove de les interfícies IPointerDownHandler, IPointerUpHandler i IPointerMoveHandler, respectivament. En la Figura 109 es veu la declaració de la classe IslandCellScript i com es mencionen les interfícies.

```
using UnityEngine.EventSystems;

Script de Unity | 3 referencias
public class IslandCellScript : MonoBehaviour, IPointerDownHandler, IPointerUpHandler, IPointerMoveHandler
```

Figura 109: Declaració de la classe IslandCellScript

Aquests mètodes inclouen com a paràmetre un objecte de tipus PointerEventData que inclou el resultat de la operació de raycasting executada pel propi motor. El mètode OnPointerMove comença cridant la funció GetPointerCoordinates per convertir aquestes dades en la coordenada dins l'illa, tal i com es veu a la Figura 110.

```
private void GetPointerCoordinates(PointerEventData eventData, out int x, out int y)
{
    Vector3 islandPoint;
    if (eventData != null)
    {
        islandPoint = eventData.pointerCurrentRaycast.worldPosition - islandScript.transform.position;
    }
    else
    {
        RaycastHit raycastHit;
        Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
        Physics.Raycast(ray, out raycastHit, 100f);
        islandPoint = raycastHit.point - islandScript.transform.position;
    }
    Vector2 mapPoint = new Vector2(IslandGenerator.mapChunkSize / 2 + islandPoint.x, IslandGenerator.mapChunkSize / 2 - islandPoint.z);
    x = Mathf.FloorToInt(mapPoint.x);
    y = Mathf.FloorToInt(mapPoint.y);
}
```

Figura 110: Mètode GetPointerCoordinates

Per evitar repetir càlculs, el mètode OnPointerMove comença comprovant si la coordenada de l'illa que apunta el ratolí és diferent a l'anterior. Si és la mateixa, es surt del mètode sense realitzar més operacions. En la Figura 111 es pot veure com es gestiona la condició.

```
public void OnPointerMove(PointerEventData eventData)
{
    int x, y;
    GetPointerCoordinates(eventData, out x, out y);

    if ((wasOtherCellHovered || selectMode != SelectMode.None) && hoveredCell.x == x && hoveredCell.y == y) return;
```

Figura 111: Inici del mètode OnPointerMove

També interessa monitoritzar el canvi en la posició del ratolí quan es mou la càmera amb els botons del teclat. Per fer-ho, es comprova en el mètode Update si l'input horitzontal o vertical és diferent a 0. Si és el cas, es crida el mètode OnPointerMove passant per paràmetre un valor nul, i quan aquest cridi la funció GetPointerCoordinates es realitzarà el càlcul de raycast necessari per continuar. En la Figura 112es mostra l'inici del mètode Update.

```
private void Update()
{
    if (Input.GetAxis("Horizontal") != 0 || Input.GetAxis("Vertical") != 0)
    {
        OnPointerMove(null);
    }
```

Figura 112: Inici del mètode Update

El mètode CreateCell (Figura 113) s'encarrega de crear els GameObject amb els components bàsics per poder mostrar el mesh de cada cel·la. Crida el mètode GenerateCell (Figura 114), que obté els vèrtexs de la coordenada de l'illa i augmenta lleugerament el seu valor en l'eix y perquè sobresurtin del terra. Els objectes cel·la creats i les coordenades es guardaran en una matriu bidimensional.

```
private void CreateCell(Vector2 position, Material cellMaterial)
{
    GameObject newCell = new GameObject("cell");
    MeshRenderer meshRenderer = newCell.AddComponent<MeshRenderer>();
    MeshFilter meshFilter = newCell.AddComponent<MeshFilter>();

    meshRenderer.material = cellMaterial;

    MeshData cellMeshData = MeshGenerator.GenerateCell(position, 0.02f, meshData);
    Mesh mesh = cellMeshData.CreateMesh();
    meshFilter.mesh = mesh;

    newCell.transform.parent = gameManagerScript.cellsTransform;
    newCell.transform.position = transform.position;

    cells[(int)position.x, (int)position.y] = newCell;
}
```

Figura 113: Mètode CreateCell

```
public static MeshData GenerateCell(Vector2 pos, float heightOffset, MeshData islandMeshData)
{
    MeshData meshData = new MeshData(2, 2);
    int[] triangles = new int[6];
    int j = (int)(pos.y * (IslandGenerator.mapChunkSize-1) + pos.x) * 6;
    Vector3[] vertices = new Vector3[6];
    for (int i = 0; i < 6; i++) {
        vertices[i] = islandMeshData.vertices[islandMeshData.triangles[j]] + new Vector3(0, heightOffset, 0);
        j++;
    }

    meshData.vertices = new Vector3[] { vertices[0], vertices[5], vertices[2], vertices[1] };
    meshData.AddTriangle(0, 3, 2);
    meshData.AddTriangle(3, 0, 1);
    meshData.uvs = new Vector2[] { new Vector2(0, 0), new Vector2(1, 0), new Vector2(0, 1), new Vector2(1, 1) };

    return meshData;
}
```

Figura 114: Mètode GenerateCell

La lògica del sistema de cel·les depèn de l'acció que es vol dur a terme i el mode de selecció. Les accions són "col·locar un edifici", "crear un tancat", "plantar arbres", "marcar elements com a pendents d'eliminar" o "cancel·lar l'eliminació d'elements". Els mode de selecció són "cap", "seleccionant" i "edificant". En la Figura 115 es veu la declaració de les enumeracions.

```
25 referencias
public enum SelectAction { CreateEnclosure, PlaceBuilding, PlantTrees, ClearItems, CancelItemClearing };
10 referencias
private enum SelectMode { None, Selecting, Building };
```

Figura 115: Declaració de les enumeracions d'acció i mode de selecció

En el mode "cap", que serà el cas mentre el ratolí no s'està clicant, es crearà una cel·la de tipus "flotant" per allà on passi. Cada cop que canviï de cel·la l'anterior s'eliminarà. Només apareixeran cel·les en terreny edificable, és a dir, amb gespa. A la Figura 116 es pot apreciar la lògica.

```
DestroyAllCells();
hoveredCell = new Vector2(x, y);

if (selectMode == SelectMode.None)
{
    if (islandGenerator.regions[islandScript.regionMap[x, y]].type == Terrain.TerrainType.Field
        || islandGenerator.regions[islandScript.regionMap[x, y]].type == Terrain.TerrainType.Hill)
    {
        CreateCell(hoveredCell, ResourceScript.Instance.hoverMaterial);
        wasOtherCellHovered = true;

        thudSource.Play();
    }
    return;
}
```

Figura 116: Fragment del mètode OnPointerMove relatiu al mode de selecció

Quan el jugador faci clic, si el terreny no és editable es sortirà del mode d'edició. Sinó es seleccionarà la cel·la on es trobi i s'entrarà en el mode "seleccionant". En aquest estat, quan es canviï de cel·la es calcularà un rectangle que prendrà com a cantonades la primera cel·la seleccionada i l'actual, i totes les cel·les que es trobin dins també es comptaran com a seleccionades. Si la selecció actual no és possible quedarà marcada com a invàlida. La figura Figura 117 mostra dues seleccions, una de vàlida i una d'invàlida.

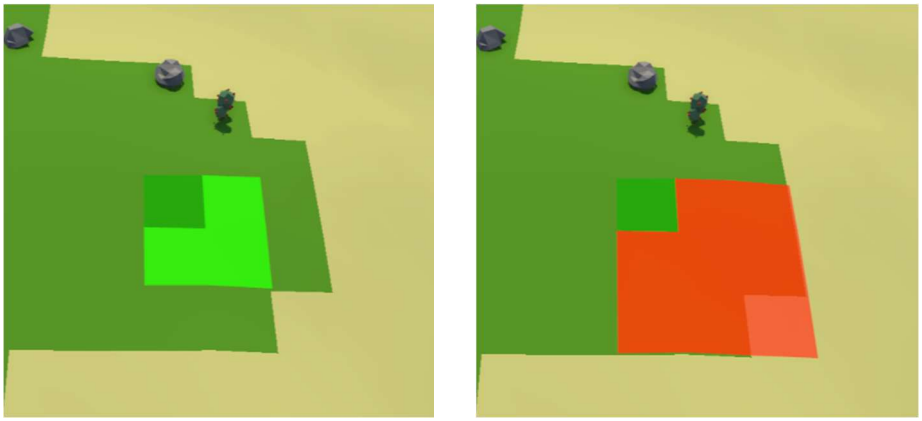


Figura 117: Una selecció vàlida (esquerra) i una d'invàlida (dreta).

La validesa d'una selecció es basa en l'acció que es vulgui dur a terme. En qualsevol dels casos, una selecció és invàlida quan la zona inclou un tipus de terreny no edificable, com sorra, o es solapa amb les cel·les d'una construcció existent. També es tenen en compte els elements de l'illa a l'hora de decidir la validesa de la selecció, així que s'afegeixen en una llista per més endavant. A la Figura 118 es veu el fragment de codi que ho executa.

```
int index = 0;
for (int row = (int)startCell.y; row <= endCell.y; row++)
{
    for (int col = (int)startCell.x; col <= endCell.x; col++)
    {
        selectedCells[index] = new Vector2(col, row);
        if (islandScript.regionMap[col, row] < 0 // No està disponible
            || (islandGenerator.regions[islandScript.regionMap[col, row]].type != Terrain.TerrainType.Field
            && islandGenerator.regions[islandScript.regionMap[col, row]].type != Terrain.TerrainType.Hill))
        {
            isSelectionValid = false;
        }
        else if (islandScript.isCellTaken(selectedCells[index]))
        {
            selectedItems.Add(islandScript.itemDictionary[selectedCells[index]]);
        }
        index++;
    }
}
```

Figura 118: Fragment del mètode OnPointerMove relatiu als modes "seleccionant" i "edificant"

Si la funció és "marcar elements com a pendents d'eliminar" o "cancel·lar l'eliminació d'elements", la selecció serà invàlida si no inclou cap element. En qualsevol altre cas la lògica és invertida. En la Figura 119es mostra aquesta lògica.

```
if (selectAction == SelectAction.ClearItems || selectAction == SelectAction.CancelItemClearing)
{
    if (selectedItems.Count == 0) isSelectionValid = false;
}
```

Figura 119: Fragment del mètode OnPointerMove relatiu a les funcions "marcar elements com a pendents d'eliminar" i "cancel·lar l'eliminació d'elements"

A més, si s'està construint un tancat, la selecció serà invàlida si l'amplada o la llargada són inferiors a 3 (la mida mínima) o no es tenen suficients materials per construir el nombre de tanques necessàries. A la Figura 120 es veu el cas mencionat.

```
else if (selectAction == SelectAction.CreateEnclosure)
{
    int availableWood = islandScript.GetResourceAmount(ResourceScript.ResourceType.Material, (int)ResourceScript.MaterialType.Wood);
    int fenceAmount = (int)((endCell.x - startCell.x) * 2 + (endCell.y - startCell.y - 2) * 2);
    if (selectedItems.Count > 0 || endCell.x - startCell.x < 2 || endCell.y - startCell.y < 2
        || availableWood < fenceAmount)
    {
        isSelectionValid = false;
    }
}
```

Figura 120: Fragment del mètode OnPointerMove relatiu a la funció "crear un tancat"

En la funció "col·locar un edifici", una instància de l'edifici seleccionat acompanyarà el ratolí i es posicionarà al terreny. Per evitar que l'edifici estigui flotant o enterrat es crida el mètode GetBuildingPosition, que fa una mitjana de les altituds de les cel·les que ocupa. A la Figura 121 i la Figura 122 es poden veure els dos mètodes.

```
else if (selectAction == SelectAction.PlaceBuilding)
{
    if (selectedItems.Count > 0) isSelectionValid = false;

    //s'obté la posició de l'edifici segons el vèrtex inicial i l'alçada mitjana de la zona
    Vector3 vertex = MeshGenerator.GetBuildingPosition(selectedCells, hoveredCell, vertexIndex, islandScript.meshData);
    selectedBuilding.transform.position = islandScript.transform.position + vertex;
    ChangeSelectedBuildingColor(isSelectionValid ? Color.gray : Color.red);
}
```

Figura 121: Fragment del mètode OnPointerMove relatiu a la funció "col·locar un edifici"

```
public static Vector3 GetBuildingPosition(Vector2[] selectedCells, Vector2 hoveredCell, int vertexIndex, MeshData islandMeshData)
{
    float heightSum = 0;
    foreach(Vector2 cell in selectedCells)
    {
        Vector3[] cellVertices = GetCellVertices(hoveredCell, islandMeshData);
        heightSum += (cellVertices[0].y + cellVertices[1].y + cellVertices[2].y + cellVertices[3].y) / 4;
    }
    Vector3 vertex = GetCellVertices(hoveredCell, islandMeshData)[vertexIndex];
    return new Vector3(vertex.x, heightSum / selectedCells.Length, vertex.z);
}
```

Figura 122: Mètode GetBuildingPosition de MeshGenerator

La Figura 123 mostra dues possibles ubicacions d'un edifici, una de vàlida i una d'invàlida.

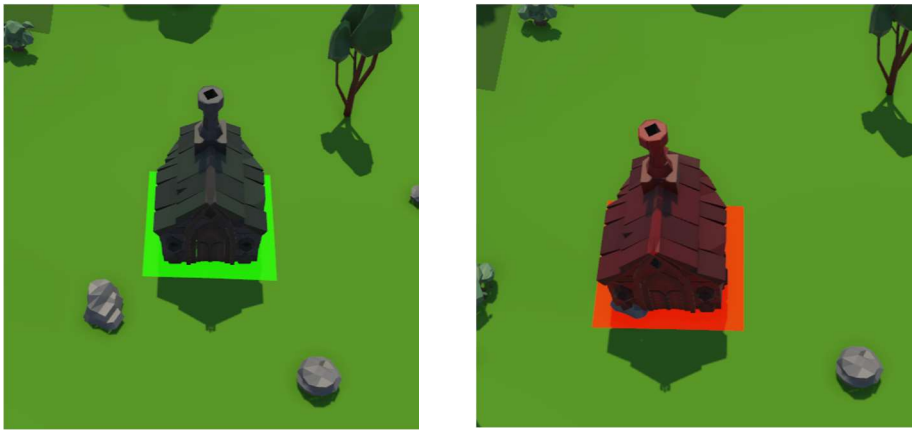


Figura 123: Una posició vàlida per un edifici (esquerra) i una d'invàlida (dreta).

Com es pot veure a la Figura 124, en el mètode Update es capturen les tecles punt i coma per saber si es vol rotar l'edifici en sentit horari i antihorari, respectivament.

```
if (selectMode == SelectMode.Building) //Si està col·locant un edifici
{
    if (Input.GetKeyDown(KeyCode.Comma))
    {
        selectedBuilding.transform.Rotate(new Vector3(0, -90, 0)); //Es rota en sentit antihorari
        selectedBuilding.orientation -= 1;
        if (selectedBuilding.orientation == -1) selectedBuilding.orientation = 3;
        OnPointerMove(null);
    }
    else if (Input.GetKeyDown(KeyCode.Period))
    {
        selectedBuilding.transform.Rotate(new Vector3(0, 90, 0)); //Es rota en sentit horari
        selectedBuilding.orientation = (selectedBuilding.orientation + 1) % 4;
        OnPointerMove(null);
    }
```

Figura 124: Fragment del mètode Update relatiu al mode "edificant"

Quan es conclou la selecció es destruiran totes les cel·les. Si la zona era vàlida s'efectuarà l'acció que ha triat anteriorment. En les funcions "col·locar un edifici" i "crear un tancat", la construcció afegida quedarà seleccionada, es restaran els recursos que toquin i apareixeran els botons per editar-la. En la Figura 125 es pot veure aquesta lògica.

```
ChangeSelectedBuildingColor(Color.white);
selectedBuilding.cells = selectedCells;
selectedBuilding.position = selectedBuilding.transform.position;
islandScript.UseResource(ResourceScript.ResourceType.Material, (int)ResourceScript.MaterialType.Wood, selectedBuilding.requiredWood);
islandScript.UseResource(ResourceScript.ResourceType.Material, (int)ResourceScript.MaterialType.Stone, selectedBuilding.requiredStone);
islandScript.AddConstruction(selectedBuilding); //Col·locar edifici
GameManager.Instance.islandSelectionScript.enabled = true;
GameManager.Instance.islandSelectionScript.SelectConstruction(selectedBuilding);

Instantiate(puffPrefab, selectedBuilding.entry.position, puffPrefab.transform.rotation);
```

Figura 125: Fragment del mètode OnPointerDown relatiu a la funció "col·locar un edifici"

En canvi, quan es crea un tancat es col·loquen tanques de fusta al perímetre, a certa distància de la vora, per tal de permetre el pas als personatges. Es deixa la cantonada inferior esquerra lliure per l'entrada. A la Figura 126 i la Figura 127 es mostra el codi que ho executa.

```
if (selectAction == SelectAction.CreateEnclosure)
{
    int fenceAmount = (int)((selectedCells[selectedCells.Length - 1].x - selectedCells[0].x + 1) * 2 + (selectedCells[selectedCells.Length - 1].y - selectedCells[0].y - 1) * 2);
    islandScript.UseResource(ResourceScript.ResourceType.Material, (int)ResourceScript.MaterialType.Wood, fenceAmount);
    EnclosureScript enclosureScript = islandScript.CreateEnclosure(selectedEnclosureType, selectedCells);
    islandScript.AddConstruction(enclosureScript);
    GameManager.Instance.islandSelectionScript.enabled = true;
    GameManager.Instance.islandSelectionScript.SelectConstruction(enclosureScript);

    Instantiate(puffPrefab, enclosureScript.entry.position, puffPrefab.transform.rotation);
}
```

Figura 126: Fragment del mètode OnPointerUp relatiu a la funció "crear un tancat"

```
public virtual void InitializeEnclosure(EnclosureScript enclosureScript, IslandScript islandScript)
{
    this.islandScript = islandScript;
    constructionType = ConstructionType.Enclosure;
    width = (int)cells[cells.Length - 1].x - (int)cells[0].x + 1;
    length = (int)cells[cells.Length - 1].y - (int)cells[0].y + 1;

    GameObject fences = new GameObject("Fences");
    fences.transform.parent = transform;
    fences.transform.localPosition = Vector3.zero;
    Vector3[] positions;
    Quaternion[] rotations;
    MeshGenerator.GetFencePositions(cells, islandScript.meshData, out positions, out rotations);
    for (int i = 0; i < positions.Length - 1; i++)
    {
        GameObject fence = Instantiate(ResourceScript.Instance.GetRandomFencePrefab(), transform.position + positions[i], rotations[i], fences.transform);
        if (i == 0) minPos = fence.transform.localPosition;
        else if (i == positions.Length - 2) maxPos = fence.transform.localPosition - new Vector3(0, 0, 1);
    }

    if (enclosureType == EnclosureType.Pen)
    {
        GameObject closedGate = Instantiate(ResourceScript.Instance.gate, transform.position + positions[positions.Length - 1], rotations[rotations.Length - 1], fences.transform);
    }
    else
    {
        GameObject post = Instantiate(ResourceScript.Instance.post, transform.position + positions[positions.Length - 1], rotations[rotations.Length - 1], fences.transform);
    }
    outline = fences.AddComponent<Outline>();
    outline.enabled = false;

    BoxCollider boxCollider = gameObject.AddComponent<BoxCollider>();
    boxCollider.center = (minPos + maxPos) / 2;
    boxCollider.size = new Vector3(maxPos.x - minPos.x, 1, minPos.z - maxPos.z);
    boxCollider.isTrigger = true;
    gameObject.layer = 8;

    maxPeasants = (width - 2) * (length - 2);
    minPos += transform.position;
    maxPos += transform.position;

    entry = Instantiate(ResourceScript.Instance.enclosureCenter, transform.position + boxCollider.center, Quaternion.identity, transform).transform;
}
```

Figura 127: Mètode InitializeEnclosure

En la funció "plantar arbres" es col·locarà una llavor d'arbre en cada cel·la seleccionada. Al cap d'un temps creixerà un arbre en aquella posició i del tipus que pertoqui segons el terreny. A la Figura 128 es pot veure la lògica.

```
else if(selectAction == SelectAction.PlantTrees)
{
    islandScript.UseResource(ResourceScript.ResourceType.Material, (int)ResourceScript.MaterialType.Sprout, selectedCells.Length);
    islandScript.PlantTrees(selectedCells);
    CanvasScript.Instance.ShowDefaultButtons();
}
```

Figura 128: Fragment del mètode OnPointerUp relatiu la funció "plantar arbres"

En la funció "marcar elements com a pendents d'eliminar", els elements seleccionats tindran una silueta vermella que indica l'estat. Finalment, en la funció "cancel·lar l'eliminació d'elements" els elements seleccionats que estiguessin marcats deixaran d'estar-ho. En ambdós casos es crida el mètode ChangeItemClearingState, que s'encarrega de la gestió. La Figura 129 i la Figura 130 mostren el codi que ho executa.

```
else
{
    foreach (ItemScript item in selectedItems)
    {
        item.ChangeItemClearingState(selectAction == SelectAction.ClearItems);
    }
    CanvasScript.Instance.ShowDefaultButtons();
}
```

Figura 129: Fragment del mètode OnPointerUp relatiu a les funcions "marcar elements com a pendents d'eliminar" i "cancel·lar l'eliminació d'elements"

```
public void ChangeItemClearingState(bool toClear)
{
    if (isScheduledForClearing == toClear) return;

    isScheduledForClearing = toClear;
    outline.enabled = toClear;
    if (toClear) islandScript.AddItemToClear(this);
    else islandScript.RemoveItemToClear(this);
}
```

Figura 130: Mètode ChangeItemClearingState

Els elements marcats com a pendents de treure tenen un contorn vermell per indicar el seu estat, com es pot veure a la Figura 131.

Figura 131: Elements marcats com a pendents de treure.

6.4. Personatges

6.4.1. Model i animacions

El paquet gratuït "Lowpoly Medieval Peasants" de la Unity Asset Store conté 5 models humans amb parts intercanviables, que es poden veure en la Figura 132. Tots ells tenen esquelet i són aptes per les animacions de Mixamo. Addicionalment utilitzen un sol material comú basat en un shader propi que el fa completament personalitzable. Pel projecte es necessitaven el model del nen, el de la dona i el de l'home. Per tant, es compta amb 3 bases amb milers de possibles variants.

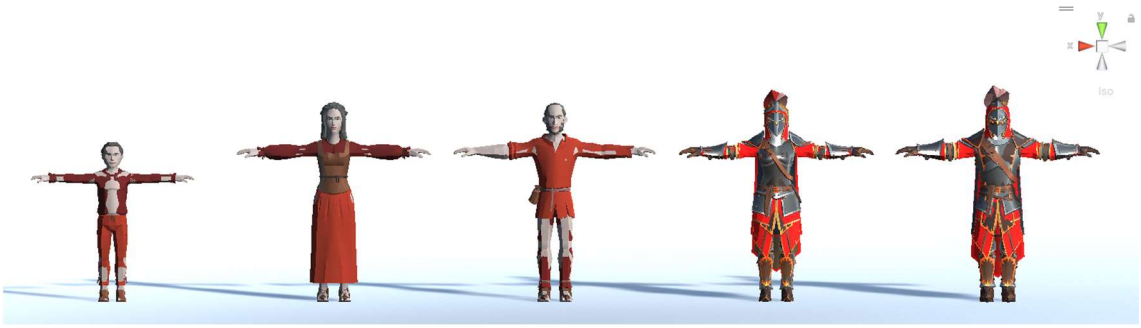


Figura 132: Models 3D base

Quan un personatge és instanciat les característiques de la variant són escollides aleatòriament, basant-se en alguns paràmetres. Pot tenir un de dos possibles caps i, si és home, diferents tipus de barbes. El color del cabell i de la pell, a diferència del color de la roba, tindrà només alguns valors possibles. No s'ha considerat necessari canviar el color de les sabates, cordes i bosses. A la Figura 133 es mostra el codi PeasantScript des de l'inspector i la Figura 134 es mostra la funció SetAppearence.

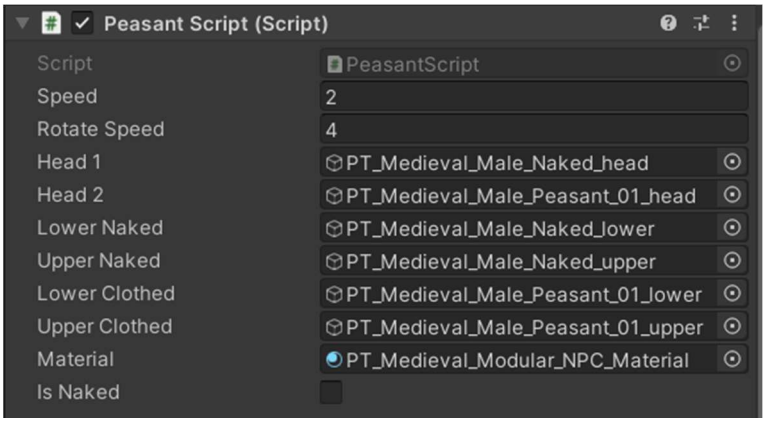


Figura 133: Inspector del prefab del personatge

```
public void SetAppearence()
{
    if (Random.Range(0, 2) == 0)
    {
        head1.SetActive(false);
        head = head2;
    }
    else
    {
        head2.SetActive(false);
        head = head1;
    }

    if (isNaked)
    {
        lowerClothed.SetActive(false);
        upperClothed.SetActive(false);
        lower = lowerNaked;
        upper = upperNaked;
    }
    else
    {
        lowerNaked.SetActive(false);
        upperNaked.SetActive(false);
        lower = lowerClothed;
        upper = upperClothed;
    }

    Material newMaterial = new Material(material);
    newMaterial.SetColor("_OTHERCOLOR", Random.ColorHSV()); //Top roba interior dona
    //newMaterial.SetColor("_LEATHER1COLOR", Random.ColorHSV()); //Sabates home i dona
    //newMaterial.SetColor("_LEATHER2COLOR", Random.ColorHSV()); //Bossa, armilla, cordes i sabates
    //newMaterial.SetColor("_LEATHER3COLOR", Random.ColorHSV()); //Cordes dona
    //newMaterial.SetColor("_LEATHER4COLOR", Random.ColorHSV()); //Cordes home
    newMaterial.SetColor("_CLOTH3COLOR", Random.ColorHSV()); //Pantalons, blusa, samarreta
    newMaterial.SetColor("_CLOTH4COLOR", Random.ColorHSV()); //Camisa, faldilla, pantalons + roba interior
    newMaterial.SetColor("_SKINCOLOR", NPCManager.GetRandomSkinColor()); //Color pell
    newMaterial.SetColor("_HAIRCOLOR", NPCManager.GetRandomHairColor()); //Color cabell
    head.GetComponent<SkinnedMeshRenderer>().material = newMaterial;
    lower.GetComponent<SkinnedMeshRenderer>().material = newMaterial;
    upper.GetComponent<SkinnedMeshRenderer>().material = newMaterial;
}
```

Figura 134: Mètode SetAppearence de PeasantScript, que estableix l'aparença del model

El següent pas és obtenir les animacions de Mixamo. S'ha creat un controlador d'animacions bàsic pel model YBot que seguidament s'adjudica als personatges. L'única cosa que s'ha de tenir en compte per poder reutilitzar les animacions és establir el tipus de rig "humanoide". En la Figura 135 es mostra el controlador d'animacions dels personatges adults.

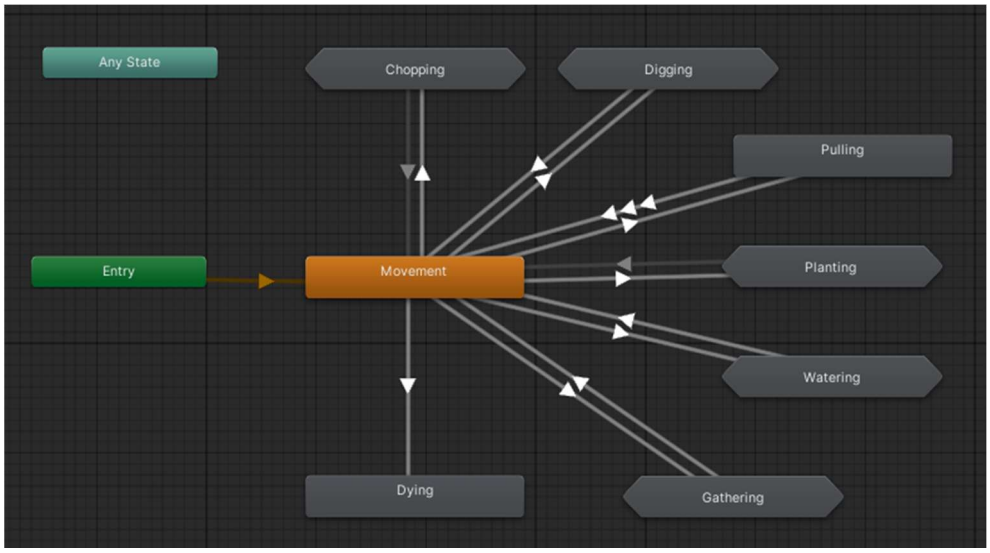


Figura 135: Controlador d'animacions dels personatges adults

L'animador té com a animació per defecte un BlendTree dedicada al simple moviment. Segons el valor de la variable Speed el personatge realitzarà l'animació de caminar, córrer o estar quiet. A la Figura 136 es veu el BlendTree anomenat Movement.

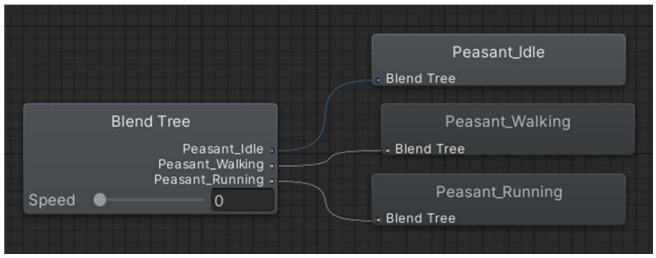


Figura 136: BlendTree Movement

Els personatges adults tenen objectes com destrals, que han d'aparèixer i desaparèixer en certs instants de l'animació. És per això que s'utilitza el sistema d'esdeveniments de les animacions, que permet escollir el mètode a cridar en cada fotograma de l'animació. Cal mencionar que els esdeveniments no són cridats si es troben en el primer o últim fotograma, el qual es sospita que té a veure amb la unió entre clips d'animació. A la Figura 137 es veu l'animació de desenfundar la destral, que en un fotograma té un esdeveniment que crida el mètode ToggleAxe.

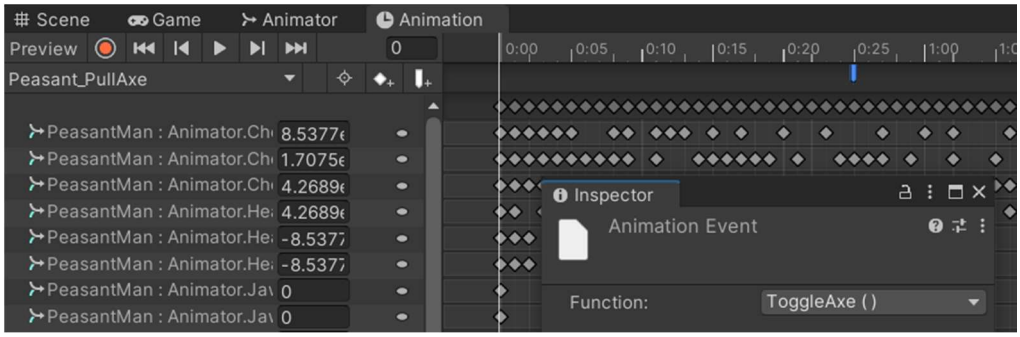


Figura 137: Animació de desenfundar la destral

En la Figura 138 es mostren uns quants personatges caminant per un espai.

Figura 138: Personatges caminant

6.4.2. Rutines

El mètode Update de PeasantScript crida el mètode UpdateDetails (Figura 139), que controla els diversos atributs del personatge. A cada actualització augmenta els nivells de gana, cansament i edat del personatge una quantitat preestablerta. Si el nivell de gana és supera el límit, busca la taverna més propera. Si el nivell de cansament supera el límit, busca la casa més propera. En qualsevol dels dos casos crida la funció abstracta UpdateTask, que veurem més endavant.

```
protected void UpdateDetails()
{
    if (hunger < 1)
    {
        hunger += Time.deltaTime * hungerSpeed;
    }
    else if (tavern == null)
    {
        hunger = 1;
        speechBubble.IsHungry();
        tavern = (TavernScript)islandScript.GetAvailableBuilding(BuildingScript.BuildingType.Tavern, this);
        if (tavern != null)
        {
            tavern.AddPeasant(this);
            UpdateTask();
        }
    }

    if (exhaustion < 1)
    {
        exhaustion += Time.deltaTime * exhaustionSpeed;
    }
    else if (cabin == null)
    {
        exhaustion = 1;
        speechBubble.IsTired();
        cabin = (CabinScript)islandScript.GetAvailableBuilding(BuildingScript.BuildingType.Cabin, this);
        if (cabin != null)
        {
            cabin.AddPeasant(this);
            UpdateTask();
        }
    }

    age += Time.deltaTime * ageSpeed;

    peasantRowScript?.UpdatePeasantDetails();
}
```

Figura 139: Mètode UpdateDetails de PeasantScript

El mètode Update de PeasantScript també crida el mètode CheckIfArrivedAtDestination (Figura 140), que comprova si el personatge ha arribat al destí. Per estar-ne segur revisa diverses condicions: Primer, cal que no estigui dins un edifici, ja que la navegació en interiors no té importància. Segon, que el sistema de navegació estigui actiu i es trobi en una superfície navegable, per tal que els càlculs siguin impossibles. Tercer, que el sistema de navegació confirmi que ha arribat, el qual es pot garantir quan no li queda camí pendent, la distància amb el destí és inferior a la distància d'aturada i la velocitat del personatge és 0. Si es dóna el cas es crida la funció abstracta ArrivedAtDestination.

```
protected void CheckIfArrivedAtDestination()
{
    if (!isInBuilding && navMeshAgent.isActiveAndEnabled
        && navMeshAgent.isOnNavMesh && !navMeshAgent.pathPending
        && navMeshAgent.remainingDistance <= navMeshAgent.stoppingDistance
        && (!navMeshAgent.hasPath || navMeshAgent.velocity.sqrMagnitude == 0f))
    {
        animator.SetFloat("Speed", 0);
        ArrivedAtDestination();
    }
}
```

Figura 140: Mètode CheckIfArrivedAtDestination de PeasantScript

UpdateTask és un mètode que decideix el nou destí del personatge segons l'estat actual. La funció és cridada cada cop que hi ha un canvi d'estat. Com que el comportament del personatge adult és més complex que el jove, la implementació del mètode es fa en ambdues classes PeasantAdultScript i PeasantChildScript.

El mètode UpdateTask de la classe PeasantChildScript (Figura 141) és bastant senzilla, ja que la rutina d'un nen no inclourà tasques. L'assignació de destí només depèn de si ha d'anar a una construcció, que només pot ser una taverna, una casa o el vaixell. En cas contrari, la futura posició serà decidida de manera aleatòria.

```
public override void UpdateTask()
{
    StopCharacter();
    if (tavern != null) //Si ha d'anar a menjar
    {
        SetDestination(tavern.entry.position);
    }
    else if (cabin != null) //Si ha d'anar a dormir
    {
        SetDestination(cabin.entry.position);
    }
    else
    {
        if (constructionScript != null) //Si té el vaixell com a destí
        {
            SetDestination(constructionScript.entry.position);
        }
        else //Si no té destí
        {
            SetDestination(NPCManager.GetRandomPoint(transform.position));
        }
    }
}
```

Figura 141: Mètode UpdateTask de PeasantChildScript

El mètode UpdateTask de la classe PeasantAdultScript (Figura 142) té una implementació del mètode similar, però inclou tasques i permet anar a més tipus de construccions. Si té una tasca pendent anirà a realitzar-la. Si la construcció on ha d'anar és un tancat i no té tasca, escollirà una posició del recinte de manera aleatòria.

```
public override void UpdateTask()
{
    StopCharacter();
    if (tavern != null) //Si ha d'anar a menjar
    {
        SetDestination(tavern.entry.position);
    }
    else if (cabin != null) //Si ha d'anar a dormir
    {
        SetDestination(cabin.entry.position);
    }
    else if (task != null) //Si té una tasca encarregada
    {
        SetDestination(task.center);
    }
    else if (constructionScript != null) //Si té una construcció com a destí
    {
        if (constructionScript.constructionType == ConstructionScript.ConstructionType.Enclosure) //Si la construcció és exterior
        {
            SetDestination(NPCManager.GetRandomPointWithinRange(((EnclosureScript)constructionScript).minPos, ((EnclosureScript)constructionScript).maxPos));
        }
        else
        {
            SetDestination(constructionScript.entry.position);
        }
    }
    else
    {
        SetDestination(NPCManager.GetRandomPoint(transform.position));
    }
}
```

Figura 142: Mètode UpdateTask de PeasantAdultScript

El mètode ArrivedAtDestination fa que el personatge realitzi una acció segons el destí que tenia fixat. De la mateixa manera que UpdateTask, també està implementada de formes diferents en les classes PeasantAdultScript i PeasantChildScript degut a la complexitat de cada una.

A PeasantChildScript, com s'ha comentat anteriorment, només pot tenir com a destí una taverna, una casa, el vaixell i una posició aleatòria. Quan arribi a una construcció se li comunicarà a aquesta per tal que executi la lògica pròpia. En canvi, quan arribi a una posició aleatòria esperarà a rebre la següent. A més, en aquesta funció es comprovarà si el personatge ha arribat a la majoria d'edat i per tant s'ha de convertir en un adult. Si és el cas es crida el mètode AgeUpPeasant. Aquest instancia un personatge adult amb el màxim de característiques idèntiques i elimina el personatge jove. A la Figura 143 i la Figura 144 es poden veure els dos mètodes mencionats.

```
protected override void ArrivedAtDestination()
{
    if (age > 18)
    {
        AgeUpPeasant();
        return;
    }

    if (tavern != null) //Si ha anat a menjar
    {
        tavern.PeasantHasArrived(this);
        peasantRowScript?.PeasantArrivedToBuilding();
    }
    else if (cabin != null) //Si ha anat a dormir
    {
        cabin.PeasantHasArrived(this);
        peasantRowScript?.PeasantArrivedToBuilding();
    }
    else if (constructionScript != null) //Si té el vaixell com a destí
    {
        constructionScript.PeasantHasArrived(this);
        peasantRowScript?.PeasantArrivedToBuilding(true);
    }
    else
    {
        StartCoroutine(WaitForNextRandomDestination());
    }
}
```

Figura 143: Mètode ArrivedAtDestination de PeasantChildScript

```
PeasantAdultScript peasantAdultScript = Instantiate(ResourceScript.Instance.GetPeasantPrefab(PeasantType.Adult, peasantGender),
    pos, transform.rotation, transform.parent).GetComponent<PeasantAdultScript>();

peasantAdultScript.islandScript = islandScript;
peasantAdultScript.isNative = false;
peasantAdultScript.headType = Random.Range(0, 2);
peasantAdultScript._SKINCOLOR = _SKINCOLOR;
peasantAdultScript._HAIRCOLOR = _HAIRCOLOR;
peasantAdultScript._CLOTH3COLOR = _CLOTH3COLOR;
peasantAdultScript._CLOTH4COLOR = _CLOTH4COLOR;
peasantAdultScript._OTHERCOLOR = _OTHERCOLOR;

peasantRowScript?.SetPeasant(peasantAdultScript);

islandScript.peasantList.Add(peasantAdultScript);
if (constructionScript != null)
{
    constructionScript.peasantList.Add(peasantAdultScript);
    peasantAdultScript.UpdateTask();
}
else islandScript.GetNextPendingTask(peasantAdultScript);

Destroy(gameObject);
```

Figura 144: Fragment del mètode AgeUpPeasant

A PeasantAdultScript la implementació del mètode també és semblant al de la versió de PeasantChildScript, però afegint les tasques i els nous tipus de construcció. Si té una tasca pendent la realitzarà amb el mètode DoTask, que s'explica més endavant. Si arriba al seu destí dins un tancat i no té tasca triarà una nova posició dins el recinte on anar a continuació. Finalment, si el personatge ha arribat a certa edat es morirà amb el mètode Die. A la Figura 145 i la Figura 146 es poden veure dos dels mètodes mencionats.

```
protected override void ArrivedAtDestination()
{
    if (age >= lifeExpectancy)
    {
        StartCoroutine(Die());
        return;
    }

    if (tavern != null) //Si ha anat a menjar
    {
        tavern.PeasantHasArrived(this);
        peasantRowScript?.PeasantArrivedToBuilding();
    }
    else if (cabin != null) //Si ha anat a dormir
    {
        cabin.PeasantHasArrived(this);
        peasantRowScript?.PeasantArrivedToBuilding();
    }
    else if (task != null) //Si té una tasca encarregada
    {
        DoTask();
    }
    else if (constructionScript != null) //Si té una construcció com a destí
    {
        if (constructionScript.constructionType == ConstructionScript.ConstructionType.Enclosure) //Si la construcció és exterior
        {
            StartCoroutine(WaitForNextRandomDestinationInEnclosure());
        }
        else
        {
            constructionScript.PeasantHasArrived(this);
            peasantRowScript?.PeasantArrivedToBuilding(constructionScript.constructionType == ConstructionScript.ConstructionType.Ship);
        }
    }
    else
    {
        StartCoroutine(WaitForNextRandomDestination()); //Sinó esperar al següent destí
    }
}
```

Figura 145: Mètode ArrivedAtDestination de PeasantAdultScript

```
navMeshAgent.isStopped = true;
animator.SetFloat("Speed", 0);
animator.SetInteger("State", (int)PeasantAction.Dying);
yield return new WaitForSeconds(animator.GetCurrentAnimatorClipInfo(0)[0].clip.length);
Destroy(gameObject);
```

Figura 146: Fragment del mètode Die

6.4.3. Tasques

Hi ha dos tipus de tasques, cada un amb una classe pròpia: element de l'illa amb ItemScript i parcel·la d'hort amb PatchScript. Ambdues hereten de TaskScript (Figura 147), que conté la lògica comuna. Els atributs d'aquesta són el tipus, el centre, la cel·la, la font de tasques de la que forma part, el personatge que està assignat a encarregar-se'n i si se n'està encarregant actualment.

```
public abstract class TaskScript : MonoBehaviour
{
    5 referencias
    public enum TaskType { Item, Patch }
    [JsonProperty] public TaskType taskType;
    [JsonProperty] [JsonConverter(typeof(VectorConverter))] public Vector3 center;
    public Vector2 cell;

    public TaskSourceInterface taskSourceScript;
    public PeasantAdultScript peasantAdultScript;
    [JsonProperty] public bool isBeingTakenCareOf;

    4 referencias
    public abstract void TaskProgress();

    3 referencias
    public abstract void CancelTask();
}
```

Figura 147: Classe TaskScript

Les fonts de tasques són les illes i els horts, que tenen IslandScript i GardenScript respectivament. Aquestes classes implementen el mètode GetNextPendingTask de TaskSourceInterface (Figura 148), que intenta donar una nova tasca al personatge i retorna cert si ho ha aconseguit.

```
public interface TaskSourceInterface
{
    12 referencias
    public abstract bool GetNextPendingTask(PeasantAdultScript peasantAdultScript);
}
```

Figura 148: Interfície TaskSourceInterface

En les illes, les tasques són els elements pendents d'eliminar, que tenen un ItemScript (Figura 149). Els elements poden ser eliminats d'una de 4 maneres possibles, que són "tallar", "cavar", "tibar" i "recollir".

```
public class ItemScript : TaskScript
{
    5 referencias
    public enum ActionType { Chop, Dig, Pull, Pick }
    public ActionType actionType;
```

Figura 149: Classe ItemScript

Quan es busca el següent element es fa un recorregut per tota la llista i s'intenta obtenir el més proper. Si s'arriba al final s'entén que no n'hi ha cap de disponible. Es considera un element com a disponible si no té cap personatge assignat i si hi pot arribar segons la NavMesh. Per exemple, una illa pot estar dividida per un riu, el qual faria impossible anar d'un cantó a l'altre. A la Figura 150 es veu el mètode GetNextPendingTask implementat a IslandScript.

```
public bool GetNextPendingTask(PeasantAdultScript peasantAdultScript)
{
    if (!peasantAdultScript.CanBeAsignedTask())
    {
        peasantAdultScript.UpdateTask();
        return false;
    }

    ItemScript closestItemScript = null;
    float minDistance = -1;
    foreach (ItemScript itemScript in itemsToClear)
    {
        float distance;
        if (itemScript.peasantAdultScript == null //Si no té un NPC vinculat
            && NPCManager.CheckIfClosest(itemScript.transform.position, peasantAdultScript.GetComponent<NavMeshAgent>(), minDistance, out distance))
        {
            closestItemScript = itemScript;
            minDistance = distance;
        }
    }
    peasantAdultScript.AssignTask(closestItemScript);
    return closestItemScript != null;
}
```

Figura 150: Mètode GetNextPendingTask de IslandScript

En els horts, les tasques són les cel·les cultivables, que tenen un PatchScript (Figura 151). El estats possibles són "plantat", "crescut", "florit", "mort" i "buit".

```
public class PatchScript : TaskScript
{
    23 referencias
    public enum CropState { Planted, Grown, Blossomed, Dead, Barren }

    private GardenScript gardenScript;
    [JsonProperty] public ResourceScript.CropType cropType;
    [JsonProperty] public CropState cropState = CropState.Barren;
```

Figura 151: Classe PatchScript

Es guarda en la variable lastWorkedOnPatch l'índex de l'última cel·la de la llista a la qual se li ha assignat un personatge. Llavors, quan es busca la següent cel·la per assignar-hi algú es comença per aquella i es fa una volta fins a arribar a la mateixa. Si es completa la volta, s'entén que no s'ha trobat cap cel·la disponible. Es considera una cel·la com a disponible si no té cap personatge assignat i, en el cas que encara no tingui cap llavor plantada, si hi ha una llavor disponible. A la Figura 152 es veu el mètode GetNextPendingTask implementat a GardenScript.

```
public override bool GetNextPendingTask(PeasantAdultScript peasantAdultScript)
{
    if (!peasantAdultScript.CanBeAsignedTask())
    {
        peasantAdultScript.UpdateTask();
        return false;
    }

    PatchScript nextPatchScript = null;
    int index = (lastWorkedOnPatch + 1) % patchDictionary.Count;
    while(index != lastWorkedOnPatch)
    {
        PatchScript patchScript = patchDictionary.ElementAt(index).Value;
        if (patchScript.peasantAdultScript == null
            && !(patchScript.cropState == PatchScript.CropState.Barren
            && islandScript.GetResourceAmount(ResourceScript.ResourceType.Crop, (int)patchScript.cropType) == 0))
        {
            if(patchScript.cropState == PatchScript.CropState.Barren)
            {
                islandScript.UseResource(ResourceScript.ResourceType.Crop, (int)patchScript.cropType);
            }
            lastWorkedOnPatch = index;
            nextPatchScript = patchScript;
            break;
        }
        index = (index + 1) % patchDictionary.Count;
    }

    peasantAdultScript.AssignTask(nextPatchScript);
    return nextPatchScript != null;
}
```

Figura 152: Mètode GetNextPendingTask de GardenScript

El mètode CanBeAssignedTask de PeasantAdultScript retorna cert si se li pot assignar una tasca al personatge, que és quan no té una tasca ja assignada i té els nivells de gana i cansament per sota dels màxims. El mètode AssignTask, que serveix per assignar-li una tasca, crida el mètode UpdateTask per avisar que s'ha canviat l'estat. En la Figura 153 es mostren els dos mètodes mencionats.

```
3 referencias
public bool CanBeAsignedTask()
{
    return task == null && hunger < 1 && exhaustion < 1;
}

8 referencias
public void AssignTask(TaskScript taskScript)
{
    task = taskScript;
    if (task != null) task.peasantAdultScript = this;
    UpdateTask();
}
```

Figura 153: Mètodes CanBeAssignedTask i AssignTask

El mètode DoTask (Figura 154) influeix en el sistema d'animacions del personatge perquè realitzi l'animació associada amb la tasca en qüestió. En la següent imatge es pot veure com, segons el tipus de tasca, s'estableixen els valors de les variables del controlador.

```
public void DoTask()
{
    task.isBeingTakenCareOf = true;
    if (task.taskType == TaskScript.TaskType.Item)
    {
        switch (((ItemScript)task).actionType)
        {
            case ItemScript.ActionType.Chop: animator.SetInteger("State", (int)PeasantAction.Chopping); break;
            case ItemScript.ActionType.Dig: animator.SetInteger("State", (int)PeasantAction.Digging); break;
            case ItemScript.ActionType.Pull: animator.SetInteger("State", (int)PeasantAction.Pulling); break;
            case ItemScript.ActionType.Pick: animator.SetInteger("Pick", 0); animator.SetInteger("State", (int)PeasantAction.Gathering); break;
        }
    }
    else if (task.taskType == TaskScript.TaskType.Patch)
    {
        PatchScript patchScript = (PatchScript)task;
        if (patchScript.cropState != PatchScript.CropState.Barren &&
            patchScript.cropType != ((GardenScript)patchScript.taskSourceScript).cropDictionary[patchScript.cell]) //S'ha d'arrencar l'anterior planta
        {
            animator.SetInteger("State", (int)PeasantAction.Pulling);
        }
        else
        {
            switch (patchScript.cropState)
            {
                case PatchScript.CropState.Barren: animator.SetInteger("State", (int)PeasantAction.Planting); break;
                case PatchScript.CropState.Planted: case PatchScript.CropState.Grown: animator.SetInteger("State", (int)PeasantAction.Watering); break;
                case PatchScript.CropState.Blossomed: animator.SetInteger("Pick", 1); animator.SetInteger("State", (int)PeasantAction.Gathering); break;
            }
        }
    }
}
```

Figura 154: Mètode DoTask

En el moment que s'arribi a un fotograma concret de l'animació es cridarà el mètode TaskProgress de PeasantAdultScript, mostrat en la Figura 155, que augmentarà els nivells de gana i cansament del personatge i cridarà el mètode TaskProgress de la tasca.

```
0 referencias
private void TaskProgress()
{
    if(task != null)
    {
        task.TaskProgress();
        hunger += 0.05f;
        exhaustion += 0.05f;
    }
}
```

Figura 155: Mètode TaskProgress de PeasantAdultScript

En ItemScript, amb cada crida a TaskProgress es resta una unitat a la quantitat de material que li queda i es suma una unitat del material de l'element a l'inventari. Quan es quedi sense material s'eliminarà de la llista d'elements pendents de treure i serà destruit. En el cas que fos un arbre o una llavor d'arbre s'afegirà també una llavor d'arbre a l'inventari. En la Figura 156 es veu el mètode.

```
public override void TaskProgress()
{
    materialAmount--;
    islandScript.AddResource(ResourceScript.ResourceType.Material, (int)materialType);
    if (materialAmount == 0)
    {
        if(name.Substring(0, 4).ToLower().Equals("tree"))
        {
            islandScript.AddResource(ResourceScript.ResourceType.Material, (int)ResourceScript.MaterialType.Sprout);
        }
        islandScript.RemoveItemToClear(this);
        islandScript.RemoveItemAtCell(cell);
        Destroy(gameObject);
    }
}
```

Figura 156: Mètode TaskProgress de ItemScript

En PatchScript, en canvi, es fan canvis en la planta depenent de l'estat de la mateixa. Si calia plantar-ne una es resta una verdura del tipus establert de l'inventari. En canvi, si es recullen els fruits es sumen 3 unitats. En qualsevol cas es reinicia el comptador que indica el temps que porta la planta sense ser cuidada i se li assigna una nova planta al personatge. A la Figura 157 es mostra el mètode.

```
public override void TaskProgress()
{
    if (cropState == CropState.Dead || cropType != gardenScript.cropDictionary[cell]) // S'arrenca l'anterior planta
    {
        cropState = CropState.Barren;
        cropType = gardenScript.cropDictionary[cell];
        Destroy(crop);
    }
    else if (cropState == CropState.Barren) // Es planta una llavor
    {
        cropType = gardenScript.cropDictionary[cell];
        orientation = Random.Range(0, 359);
        cropState = CropState.Planted;
        crop = Instantiate(ResourceScript.Instance.GetCropPrefab(cropType, cropState), center, Quaternion.Euler(0, orientation, 0), transform);
        crop.transform.localScale = Vector3.one * 0.4f;
    }
    else if (cropState == CropState.Blossomed) // Es cullen els fruits
    {
        gardenScript.islandScript.AddResource(ResourceScript.ResourceType.Crop, (int)cropType, 3);
        cropState = CropState.Grown;
        ChangeCropState();
    }
    isBeingTakenCareOf = false;
    timeSinceLastTakenCareOf = 0;

    peasantAdultScript.task = null;
    taskSourceScript.GetNextPendingTask(peasantAdultScript);
    peasantAdultScript = null;
}
```

Figura 157: Mètode TaskProgress de PatchScript

En el mètode Update de PatchScript es fan un parell de comprovacions. Si ha passat massa temps des de l'últim cop que s'ha cuidat de la planta i la parcel·la no és buida, la planta es mor. Si en canvi ha passat massa temps des de l'última evolució de la planta aquesta passarà al següent estat. A la Figura 158 es pot apreciar la lògica.

```
private void Update()
{
    timeSinceLastStateChange += Time.deltaTime * gardenScript.level;
    timeSinceLastTakenCareOf += Time.deltaTime / gardenScript.level;

    if (!isBeingTakenCareOf)
    {
        if(timeSinceLastTakenCareOf > maxUnattendedTime && cropState != CropState.Barren)
        {
            cropState = CropState.Dead;
            ChangeCropState();
        }
        else if(timeSinceLastStateChange > timeBetweenStates && cropState < CropState.Blossomed)
        {
            cropState++;
            ChangeCropState();
        }
    }
}
```

Figura 158: Mètode Update de PatchScript

Si en algun moment previ a la crida del mètode TaskProgress es produeix un esdeveniment inesperat (per exemple, es desmarca un element marcat com a pendent d'eliminar), la tasca queda cancel·lada amb el mètode CancelTask. A PatchScript, si la parcel·la és buida, i per tant s'estava a punt de col·locar una llavor, es suma una unitat al tipus de verdura corresponent de l'inventari. A la Figura 159 i la Figura 160 es mostren els dos mètodes mencionats.

```
public void CancelTask()
{
    if (task != null)
    {
        animator.SetTrigger("CancelTask");
        task.CancelTask();
        task = null;
    }
}
```

Figura 159: Mètode CancelTask de PeasantAdultScript

```
public override void CancelTask()
{
    isBeingTakenCareOf = false;
    if (cropState == CropState.Barren)
    {
        ((GardenScript)taskSourceScript).islandScript.AddResource(ResourceScript.ResourceType.Crop, (int)cropType);
    }
}
```

Figura 160: Mètode CancelTask de PatchScript

6.5. Construccions

Totes les construccions hereten de la classe ConstructionScript (Figura 161). Els atributs constructionType i isService indiquen el tipus de construcció. Els atributs cells, lenght, width i maxPeasants donen informació de la posició i les dimensions. Els atributs level, peasantList i peasantsInside informen de la situació actual. Finalment, els atributs islandScript, constructionDetailsScript, entry i outline són referències a objectes que li permeten accedir-hi de forma més directa.

```
public abstract class ConstructionScript : MonoBehaviour
{
    13 referencias
    public enum ConstructionType { Ship, Enclosure, Building }
    public ConstructionType constructionType;
    public Vector2[] cells;
    public int length;
    public int width;

    public int level = 1;
    public int maxPeasants;
    public List<PeasantScript> peasantList = new List<PeasantScript>();
    public int peasantsInside;

    public IslandScript islandScript;
    public ConstructionDetailsScript constructionDetailsScript;
    public Transform entry;
    public Outline outline;
    public bool isService;
```

Figura 161: Atributs de ConstructionScript

La classe també compta amb propietats (Figura 162), que retornaran valors diferents segons el tipus de construcció. Aquestes són sprite, title, canManagePeasants, canBeRemoved, peasantCount i editorScript.

```
9 referencias
public abstract Sprite sprite { get; }
3 referencias
public virtual string title { get { return constructionType.ToString(); } }
4 referencias
public virtual bool canManagePeasants { get { return true; } }
2 referencias
public virtual bool canBeRemoved { get { return true; } }
3 referencias
public virtual int peasantCount { get { return peasantList.Count; } }

13 referencias
public abstract EditorScript editorScript { get; }
```

Figura 162: Propietats de ConstructionScript

La classe ConstructionDetailsScript s'encarrega de mostrar els detalls de la construcció seleccionada i permet al jugador fer-hi canvis a través de l'interfície. Quan es selecciona una construcció es crida el mètode SetConstructionScript (Figura 163), que segons els atributs i les propietats d'aquesta s'ompliran els camps i s'activaran o desactivaran botons.

```
public void SetConstructionDetails(ConstructionScript newConstructionScript)
{
    if (constructionScript != null) constructionScript.constructionDetailsScript = null;
    constructionScript = newConstructionScript;
    constructionScript.constructionDetailsScript = this;

    if(state == ConstructionDetailsState.Editor)
    {
        editorScript.gameObject.SetActive(false);
        editorScript = constructionScript.editorScript;
        editorScript.gameObject.SetActive(true);
        editorScript.SetEditor(constructionScript);
    }

    title.text = constructionScript.title;
    image.sprite = constructionScript.sprite;
    previousConstructionButton.interactable = nextConstructionButton.interactable =
        constructionScript.islandScript.constructionList.Count > 0;

    if (constructionScript.canManagePeasants)
    {
        peasantButtons.SetActive(true);
        peasantsOnTheirWayText.enabled = true;
    }
    else
    {
        peasantButtons.SetActive(false);
        peasantsOnTheirWayText.enabled = false;
    }
    UpdatePeasantNum();

    removeConstructionButton.enabled = constructionScript.canBeRemoved;
}
```

Figura 163: Mètode SetConstructionDetails

També s'utilitza en el mètode SwitchConstruction (Figura 164), que es crida quan es canvia de construcció a través dels botons de les fletxes. Mentre es pugui, s'iterarà per les construccions de l'illa i, quan s'arribi al final, es mostrarà el vaixell. Després es tornarà a l'inici de la llista.

```
public void SwitchConstruction(bool next)
{
    int constructionCount = constructionScript.islandScript.constructionList.Count;

    int currentConstruction = constructionScript.constructionType == ConstructionScript.ConstructionType.Ship ? constructionCount
                            : constructionScript.islandScript.constructionList.IndexOf(constructionScript);

    if (next)
    {
        currentConstruction = currentConstruction + 1;
        if (currentConstruction == constructionCount + 1) currentConstruction = 0;
    }
    else
    {
        currentConstruction -= 1;
        if (currentConstruction == -1) currentConstruction = constructionCount;
    }

    if (currentConstruction == constructionCount) GameManager.Instance.islandSelectionScript.SelectConstruction(ShipScript.Instance);
    else GameManager.Instance.islandSelectionScript.SelectConstruction(constructionScript.islandScript.constructionList[currentConstruction]);
}
```

Figura 164: Mètode SwitchConstruction

Els botons de gestionar personatges criden el mètode ManagePeasants (Figura 165). Si s'aconsegueix realitzar l'acció s'actualitzaran els números.

```
public void ManagePeasants(bool adding)
{
    if (constructionScript.islandScript.ManagePeasants(constructionScript, adding))
    {
        UpdatePeasantNum();
    }
}
```

Figura 165: Mètode ManagePeasants de ConstructionDetailsScript

L'èxit de la operació depèn del mètode ManagePeasants de IslandScript (Figura 166). Si es vol enviar un personatge a una construcció es buscarà el personatge de l'illa més proper. Si algun pot anar-hi s'eliminarà de la llista de personatges de l'illa i s'afegirà a la de la construcció. En canvi, si es vol treure un personatge de la construcció es farà el contrari.

```
public bool ManagePeasants(ConstructionScript constructionScript, bool adding)
{
    if (adding) // Enviar a la construcció
    {
        PeasantScript peasantScript = GetClosestPeasant(constructionScript.entry.position,
            constructionScript.constructionType != ConstructionScript.ConstructionType.Ship);

        if (peasantScript != null)
        {
            if(peasantScript.peasantType == PeasantScript.PeasantType.Adult)
            {
                ((PeasantAdultScript)peasantScript).CancelTask();
            }
            peasantList.Remove(peasantScript);
            constructionScript.AddPeasant(peasantScript);
            peasantScript.UpdateTask();
            return true;
        }
        return false;
    }
    else // Desvincular de la construcció
    {
        PeasantScript peasantScript = constructionScript.RemovePeasant();
        peasantList.Add(peasantScript);

        if (peasantScript.peasantType == PeasantScript.PeasantType.Adult)
        {
            GetNextPendingTask((PeasantAdultScript)peasantScript);
        }
        else peasantScript.UpdateTask();
        return true;
    }
}
```

Figura 166: Mètode ManagePeasants de IslandScript

Els mètodes AddPeasant i RemovePeasant de ConstructionScript reben o retornen els personatges a afegir o treure de la construcció, respectivament. En el segon, s'obté el primer personatge de la llista i, si es troba dins un edifici, es mou de l'interior a l'entrada del mateix amb la funció Warp de NavMeshAgent. En la Figura 167 es mostren els dos mètodes.

```
8 referencias
public virtual void AddPeasant(PeasantScript peasantScript)
{
    peasantScript.constructionScript = this;
    peasantList.Add(peasantScript);
}

12 referencias
public virtual PeasantScript RemovePeasant()
{
    PeasantScript peasantScript = peasantList[0];
    peasantList.RemoveAt(0);

    if (peasantScript.isInBuilding)
    {
        peasantsInside--;
        peasantScript.transform.parent = islandScript.npcsTransform;
        peasantScript.navMeshAgent.Warp(entry.position);
        peasantScript.isInBuilding = false;
        peasantScript.isInConstruction = false;
    }

    return peasantScript;
}
```

Figura 167: Mètodes AddPeasant i RemovePeasant

Quan un personatge arriba a l'entrada de la construcció es crida el mètode PeasantHasArrived (Figura 168), que augmenta el número de personatges a l'interior i actualitza el comptador.

```
public virtual PeasantScript PeasantHasArrived(PeasantScript peasantScript)
{
    peasantsInside++;
    UpdateConstructionDetails();
    return peasantScript;
}
```

Figura 168: Mètode PeasantHasArrived

Quan s'elimina una construcció es crida el mètode SendAllPeasantsBack (Figura 169), que utilitza RemovePeasant en bucle fins que no queden personatges a la llista.

```
public virtual void SendAllPeasantsBack()
{
    while(peasantList.Count > 0)
    {
        PeasantScript peasantScript = RemovePeasant();
        islandScript.peasantList.Add(peasantScript);
        if (peasantScript.peasantType == PeasantScript.PeasantType.Adult)
        {
            PeasantAdultScript peasantAdultScript = (PeasantAdultScript)peasantScript;
            peasantAdultScript.taskSourceInterface.GetNextPendingTask(peasantAdultScript);
        }
        else peasantScript.UpdateTask();
    }
}
```

Figura 169: Mètode SendAllPeasantsBack

La classe EnclosureScript (Figura 170) hereta de ConstructionScript i gestiona la lògica dels tancats. Afegeix a la llista d'atributs el tipus de tancat i les posicions mínima i màxima de l'interior. Es sobreescriu la propietat del títol perquè es mostri el tipus de tancat.

```
public abstract class EnclosureScript : ConstructionScript, TaskSourceInterface
{
    8 referencias
    public enum EnclosureType { Garden, Pen, Training };
    public EnclosureType enclosureType;

    public Vector3 minPos;
    public Vector3 maxPos;

    2 referencias
    public override string title { get { return enclosureType.ToString(); } }
```

Figura 170: Classe EnclosureScript

Quan un personatge entra dins els confins del tancat es detecta amb la funció OnTriggerEnter. Llavors, s'augmenta el nombre de personatges al seu interior i s'actualitza el comptador. Quan un personatge en surt i es crida OnTriggerExit, en canvi, es disminueix el número. Les dues funcions es mostren a la Figura 171.

Figura 171: Mètodes OnTriggerEnter i OnTriggerExit

La classe BuildingScript (Figura 172) hereta de ConstructionScript i gestiona la lògica dels edificis. Afegeix a la llista d'atributs el tipus d'edifici, la posició, la orientació i els recursos necessaris per construir-lo. Es sobreescriu la propietat del títol per a què es mostri el tipus d'edifici.

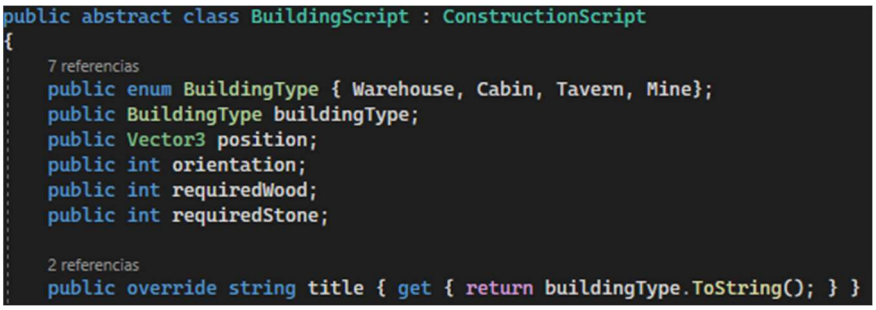


Figura 172: Classe BuildingScript

Quan un personatge arriba a l'entrada d'un edifici també es crida el mètode PeasantHasArrived (Figura 173). A diferència dels tancats, però, s'utilitza la funció Warp per moure el personatge a l'interior.

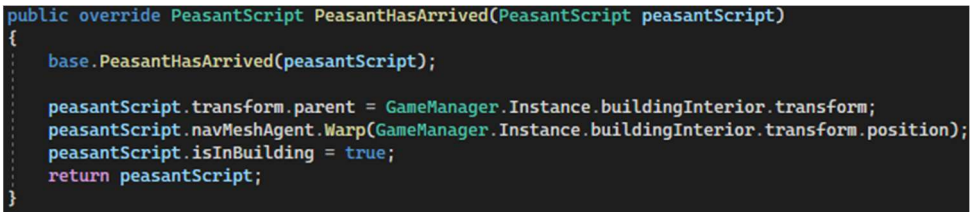


Figura 173: Mètode PeasantHasArrived de BuildingScript

6.6. Animals

Els animals són controlats per AnimalScript. Tenen alguns mètodes similars als dels personatges. Per exemple, UpdateDetails actualitza l'edat amb el temps. Si es tracta d'una cria i arriba al límit d'edat es cridarà el mètode AgeUpAnimal, que la substituirà per una versió adulta en la mateixa posició i orientació. El la Figura 174 i la Figura 175 es mostra el codi.

```
private void UpdateDetails()
{
    if (age < 1)
    {
        age += Time.deltaTime * ageSpeed;
    }
    else if ((int)animalType % 2 == 0)
    {
        age = 1;
        penScript.AgeUpAnimal(this);
    }
```

Figura 174: Fragment de UpdateDetails referent a l'edat de les cries

```
public void AgeUpAnimal(AnimalScript animalScript)
{
    AnimalType animalType = animalScript.animalType;
    AnimalScript agedUpAnimalScript = Instantiate(ResourceScript.Instance.GetAnimalPrefab(animalType + 1),
        animalScript.transform.position, animalScript.transform.rotation, animalTransform).GetComponent<AnimalScript>();
    animalList.Add(agedUpAnimalScript);
    animalList.Remove(animalScript);
    Destroy(animalScript.gameObject);

    animals[(int)animalType]--;
    animals[(int)animalType + 1]++;

    CanvasScript.Instance.UpdatePenRow(animalType);
    CanvasScript.Instance.UpdatePenRow(animalType + 1);

    agedUpAnimalScript.penScript = this;
}
```

Figura 175: Mètode AgeUpAnimal

Si en canvi es tracta d'un animal adult i arriba al límit d'edat es morirà. La corutina Die afegirà a l'inventari la quantitat de carn que pertoqui del tipus corresponent. Aquesta lògica es pot veure en la Figura 176 i la Figura 177.

```
else if (!isDying && animalToPairUpWith == null)
{
    age = 1;
    isDying = true;
    if (isConfortable)
    {
        penScript.confortableAnimals[(int)animalType].Remove(this);
    }
    StartCoroutine(Die());
}
```

Figura 176: Fragment de UpdateDetails referent a l'edat dels adults

```
public IEnumerator Die()
{
    penScript.animals[(int)animalType]--;
    penScript.animalList.Remove(this);
    navMeshAgent.isStopped = true;
    animator.SetTrigger("Death");
    yield return new WaitForSeconds(animator.GetCurrentAnimatorClipInfo(0)[0].clip.length);
    penScript.islandScript.AddResource(ResourceScript.ResourceType.Meat, (int)animalType / 2, meatAmount + penScript.level);
    Destroy(gameObject);
}
```

Figura 177: Corutina Die

En cas contrari s'augmenta el nivell de confort. Quan arribi al màxim s'afegirà a la llista d'animals còmodes amb el mètode AnimalIsConfortable, com es mostra en la Figura 178 i la Figura 179.

```
if (confortLevel < 1)
{
    confortLevel += Time.deltaTime * confortSpeed * penScript.level;
}
else if (!isConfortable)
{
    isConfortable = true;
    confortLevel = 1;
    penScript.AnimalIsConfortable(this);
}
```

Figura 178: Fragment de UpdateDetails referent al nivell de confort

```
public void AnimalIsConfortable(AnimalScript animalScript)
{
    confortableAnimals[(int)animalScript.animalType].Add(animalScript);
    ManageAnimals((int)animalScript.animalType);
}
```

Figura 179: Mètode AnimalIsConfortable

Quan això passi es cridarà el mètode ManageAnimals (Figura 180), que farà un recorregut per aquesta llista. Si troba dos animals còmodes del mateix tipus, es desitja tenir més animals d'aquest i l'aforament ho permet, els aparellarà amb el mètode GetReadyForPairing (Figura 181). Si en canvi en troba algun d'un tipus del qual interessa tenir-ne menys, el sacrificarà cridant la corutina Die.

```
private void ManageAnimals(int animalType)
{
    if (animalList.Count < maxPeasants //Hi caben més animals al tancat
        && confortableAnimals[animalType].Count >= 2 //Hi ha dos animals preparats per aparellar-se
        && desiredAmounts[animalType] >= animals[animalType]) //Es volen més animals del mateix tipus
    {
        AnimalScript animalScript1 = confortableAnimals[animalType][0];
        AnimalScript animalScript2 = confortableAnimals[animalType][1];
        confortableAnimals[animalType].RemoveAt(0);
        confortableAnimals[animalType].RemoveAt(0);

        Vector3 pairingSpot = NPCManager.GetRandomPointWithinRange(minPos, maxPos);
        animalScript1.GetReadyForPairing(pairingSpot, animalScript2);
        animalScript2.GetReadyForPairing(pairingSpot, animalScript1);

        return;
    }

    if (confortableAnimals[animalType].Count > 0
        && desiredAmounts[animalType] < animals[animalType])
    {
        AnimalScript animalScript = confortableAnimals[animalType][0];
        confortableAnimals[animalType].RemoveAt(0);
        StartCoroutine(animalScript.Die());
    }
}
```

Figura 180: Mètode ManageAnimals

```
public void GetReadyForPairing(Vector3 pairingSpot, AnimalScript animalScript)
{
    animalToPairUpWith = animalScript;
    SetDestination(pairingSpot);
}
```

Figura 181: Mètode GetReadyForPairing

A partir d'aquell moment els dos animals involucrats es dirigiran a un punt preestablert del tancat. Quan hi arribin, si són els primers s'esperaran. Si ja hi ha l'altre cridaran el mètode que els aparella. Aquesta lògica es mostra a la Figura 182.

```
private void CheckIfArrivedAtDestination()
{
    if (navMeshAgent.isActiveAndEnabled
        && navMeshAgent.isOnNavMesh && !navMeshAgent.pathPending
        && navMeshAgent.remainingDistance <= navMeshAgent.stoppingDistance
        && (!navMeshAgent.hasPath || navMeshAgent.velocity.sqrMagnitude == 0f))
    {
        animator.SetFloat("Speed", 0);
        if (animalToPairUpWith == null)
        {
            StartCoroutine(WaitForNextRandomDestination());
        }
        else
        {
            isInPlaceForPairing = true;
            if (animalToPairUpWith.isInPlaceForPairing)
            {
                penScript.BreedAnimals(this, animalToPairUpWith);
            }
        }
    }
}
```

Figura 182: Mètode CheckIfArrivedAtDestination

El mètode BreedAnimals (Figura 183) crea un número de cries del mateix tipus aleatori. També crida el mètode EndPairing (Figura 184), que reinicia el nivell de confort dels animals i els deixa voltar lliurement pel corral un cop més.

```
public void BreedAnimals(AnimalScript animalScript1, AnimalScript animalScript2)
{
    animalScript1.EndPairing();
    animalScript2.EndPairing();

    int offspringNum = Random.Range(1, level + 1);
    for(int i = 0; i < offspringNum; i++)
    {
        AnimalType animalType = animalScript1.animalType - 1;
        AnimalScript bornAnimalScript = Instantiate(Instance.GetAnimalPrefab(animalType),
            animalScript1.transform.position, animalScript1.transform.rotation, animalTransform).GetComponent<AnimalScript>();
        animalList.Add(bornAnimalScript);

        animals[(int)animalType]++;
        CanvasScript.Instance.UpdatePenRow(animalType);
        bornAnimalScript.penScript = this;
    }
}
```

Figura 183: Mètode BreedAnimals

```
public void EndPairing()
{
    isInPlaceForPairing = false;
    confortLevel = 0;
    isConfortable = false;
    animalToPairUpWith = null;
    WaitForNextRandomDestination();
}
```

Figura 184: Mètode EndPairing

6.7. Controls del vaixell

El vaixell controlat pel jugador utilitza la classe PlayerController per gestionar el moviment. El vaixell enemic, en canvi, utilitza EnemyController. Ambdues classes hereten de ShipController, que s'encarrega de convertir els diferents tipus d'entrada en desplaçament. A la Figura 185 es veu el mètode Update de la classe mare.

```
private void Update()
{
    ManageInput();
    Movement();
    Steer();
    TryToShoot();
    Balance();
    SoundAndParticles();
}
```

Figura 185: Mètode Update de ShipController

6.7.1. Moviment

El mètode ManageInput dóna valor a les variables decimals verticalInput i horizontalInput i la variable booleana wantsToShoot, que es veuen a la Figura 186.

```
protected float verticalInput;
protected float horizontalInput;
protected bool wantsToShoot;
```

Figura 186: Variables de ShipController

En la implementació del mètode a PlayerController (Figura 187), s'omplen les dues primeres amb els valors d'entrada dels eixos vertical i horitzontal, respectivament. La tercera només serà certa en els fotogrames en què el jugador premi la tecla espai. Cal remarcar que es restringeix el valor de l'eix vertical entre 0 i 1 per tal de prevenir que el vaixell es mogui enrere. A més, si el vaixell està amarrat en una illa o està pescant no es podrà moure.

```
protected override void ManageInput()
{
    if (GameManager.Instance.isInIsland || ShipScript.Instance.fishingScript.enabled)
    {
        verticalInput = 0;
        horizontalInput = 0;
        wantsToShoot = false;
        return;
    }


    verticalInput = Mathf.Clamp01(Input.GetAxis("Vertical"));
    horizontalInput = Input.GetAxis("Horizontal");
    wantsToShoot = Input.GetKeyDown(KeyCode.Space);
}
```

Figura 187: Mètode ManageInput de PlayerController

En el cas de EnemyController, el moviment del vaixell depèn de l'estat en què es troba. Aquest pot ser "comerciant", "atacant", "fugint" o "a l'espera". A la Figura 188 es mostra la declaració de l'enumeració i la variable.

```
public enum EnemyStatus { Trading, Attacking, Fleeing, StandBy }
[JsonProperty] public EnemyStatus enemyStatus = EnemyStatus.Attacking;
```

Figura 188: Variables de EnemyController

Si l'enemic està en l'estat "comerciant" o "a l'espera" es quedarà quiet, com es pot veure a la Figura 189.

```
protected override void ManageInput()
{
    if(enemyStatus == EnemyStatus.Trading || enemyStatus == EnemyStatus.StandBy)
    {
        verticalInput = 0;
        horizontalInput = 0;
        wantsToShoot = false;
        return;
    }
```

Figura 189: Fragment del mètode ManageInput de EnemyController referent als estats "comerciant" i "a l'espera"

En cas contrari, es calcularà l'angle actual amb el vaixell del jugador. Per fer-ho, s'obtindrà la diferència entre les seves coordenades dels eixos x i z. Després, es calcula l'arctangent i es converteix el resultat de radians a graus. Per a què el valor sigui fàcil d'interpretar es resta a 180 perquè es trobi entre -180 a 180. Finalment se li resta l'orientació en l'eix y del vaixell. D'aquesta manera se sap que el vaixell del jugador es troba davant quan el valor és positiu i que és darrere quan és negatiu. També se sap que el vaixell del jugador es troba a la dreta quan el valor absolut és superior a 90 i que és a l'esquerra quan és inferior. A la Figura 190 es poden veure els càlculs.

```
float xDiff = transform.position.x - ShipScript.Instance.transform.position.x;
float zDiff = transform.position.z - ShipScript.Instance.transform.position.z;
float angle = 180 - Mathf.Atan2(zDiff, xDiff) * Mathf.Rad2Deg;
float angleDiff = angle - transform.rotation.eulerAngles.y;
if (angleDiff < 0) angleDiff = 360 + angleDiff;
angleDiff = (angleDiff) % 360 - 180;

isInFront = angleDiff > 0;
isToTheRight = Mathf.Abs(angleDiff) > 90;
```

Figura 190: Fragment del mètode ManageInput de EnemyController referent al càlcul d'angle

En el cas que el vaixell enemic estigui fugint, aquest estarà sempre en moviment i virarà per mantenir el vaixell del jugador al seu darrere. És a dir, girarà a la dreta quan aquest es trobi a l'esquerra i viceversa. A la Figura 191 es veu la lògica d'aquest cas.

```
if(enemyStatus == EnemyStatus.Fleeing)
{
    verticalInput = 1;
    horizontalInput = isToTheRight ? -1 : 1;
    wantsToShoot = false;
    return;
}
```

Figura 191: Fragment del mètode ManageInput de EnemyController referent a l'estat "fugint"

En el cas que estigui atacant, es calcularà la distància amb el vaixell del jugador. Si està a una distància major a la mínima per poder disparar s'hi aproparà. Per fer-ho seguirà la mateixa lògica que si estigués fugint però virarà en sentit contrari. A la Figura 192 es pot veure el codi.

```
float distance = Vector3.Distance(transform.position, ShipScript.Instance.transform.position);
isTooFar = distance > distanceToShoot;

if (isTooFar)
{
    verticalInput = 1;
    horizontalInput = isToTheRight ? 1 : -1;
    wantsToShoot = false;
}
```

Figura 192: Fragment del mètode ManageInput de EnemyController referent al càlcul de distància

Si en canvi està prou a prop comprovarà si l'angle format amb el vaixell del jugador li permetria disparar. Si és el cas es quedarà quiet, i sinó es mourà per obtenir una millor posició. A la Figura 193 es pot veure la lògica.

```
else
{
    if (isToTheRight)
    {
        if (180 - Mathf.Abs(angleDiff) > angleToShoot)
        {
            horizontalInput = -1;
        }
        else wantsToShoot = true;
    }
    else
    {
        if (Mathf.Abs(angleDiff) > angleToShoot)
        {
            horizontalInput = 1;
        }
        else wantsToShoot = true;
    }

    verticalInput = wantsToShoot ? 0 : 1;
}
```

Figura 193: Fragment del mètode ManageInput de EnemyController a l'estat "atacant"

Per moure i girar els vaixells, se'ls aplica una força segons la variable verticalInput i un moment de rotació segons la variable horizontalInput. En els càlculs també s'utilitzen multiplicadors, que seran lleugerament majors en el vaixell del jugador per donar-li un petit avantatge. A més, els vaixells només giraran si també s'estan movent endavant, tal i com es veu en la Figura 194.

```
protected virtual void Movement()
{
    rb.AddForce(transform.forward * speed * verticalInput);
}

1 referencia
private void Steer()
{
    rb.AddTorque(transform.up * steerSpeed * horizontalInput * verticalInput);
}
```

Figura 194: Mètodes Movement i Steer de ShipController

A PlayerController, a més, es limita la posició del jugador a dins del mapa. Al mètode Start es calculen les vores del mapa a partir de la mida de l'objecte del mar i dels marges establerts. Quan es crida el mètode Movement es fa ús de Clamp per mantenir el jugador dins els límits. A la Figura 195 i la Figura 196 es pot veure el càlcul de les vores i la seva aplicació.

```
//Distància màxima amb les vores
private float xLeftMargin = 20f;
private float xRightMargin = 20f;
private float zLowerMargin = 5f;
private float zUpperMargin = 20f;

//Marges de la zona de joc
public float xLeftBounds;
public float xRightBounds;
public float zLowerBounds;
public float zUpperBounds;

Mensaje de Unity | 0 referencias
private void Start()
{
    rb = GetComponent<Rigidbody>();

    //S'obtenen les mides del mar
    GameObject sea = GameObject.Find("Sea");
    Bounds seaBounds = sea.GetComponent<Renderer>().bounds;

    //Es calculen les posicions màximes i mínimes del jugador dins el mar
    xLeftBounds = sea.transform.position.x - seaBounds.size.x / 2 + xLeftMargin;
    xRightBounds = sea.transform.position.x + seaBounds.size.x / 2 - xRightMargin;
    zLowerBounds = sea.transform.position.z - seaBounds.size.z / 2 + zLowerMargin;
    zUpperBounds = sea.transform.position.z + seaBounds.size.z / 2 - zUpperMargin;
}
```

Figura 195: Inici de la classe PlayerController

```
protected override void Movement()
{
    base.Movement();

    transform.position = new Vector3(Mathf.Clamp(transform.position.x, xLeftBounds, xRightBounds),
                                     transform.position.y,
                                     Mathf.Clamp(transform.position.z, zLowerBounds, zUpperBounds));
}
```

Figura 196: Mètode Movement de PlayerController

6.7.2. Disparar

Seguidament s'intenta disparar, el qual només passarà si en el mateix fotograma s'ha establert que el vaixell té intenció de disparar i ha passat prou temps des de l'última vegada. Si es pot disparar, es reproduirà el so dels trets i es dispararan 10 bales, 2 per cada parell de canons a banda i banda. Per cada bala s'inclourà una instància d'un ParticleSystem que representarà el fum. A la Figura 197 i la Figura 198 hi ha els mètodes que executen aquesta lògica.

```
private void TryToShoot()
{
    timeTillLastShot += Time.deltaTime;
    if (wantsToShoot && timeTillLastShot > timeBetweenShots)
    {
        soundSource.PlayOneShot(cannonClip);
        StartCoroutine(FireCannonballs());
        timeTillLastShot = 0;

        Shoot();
    }
}
```

Figura 197: Mètode TryToShoot de ShipController

```
private IEnumerator FireCannonballs()
{
    yield return new WaitForSeconds(0.1f);
    for (int i = 0; i < 5; i++)
    {
        Instantiate(leftCannonball, leftCannons[i].transform.position, leftCannonball.transform.rotation).shipController = this;
        Instantiate(smokePrefab, leftCannons[i].transform.position, Quaternion.Euler(new Vector3(0, transform.eulerAngles.y - 90, 0)));
        Instantiate(rightCannonball, rightCannons[i].transform.position, rightCannonball.transform.rotation).shipController = this;
        Instantiate(smokePrefab, rightCannons[i].transform.position, Quaternion.Euler(new Vector3(0, transform.eulerAngles.y + 90, 0)));
        yield return new WaitForSeconds(0.2f);
    }
}
```

Figura 198: Mètode FireCannonballs de ShipController

A PlayerController s'aprofita el moment en què es dispara per comprovar per si l'enemic estava comerciant. Si és el cas es canvia el seu estat a un de dos possibles: "atacant" o "fugint". A la Figura 199 es pot veure aquest canvi.

```
protected override void Shoot()
{
    if(EnemyController.Instance.enemyStatus == EnemyController.EnemyStatus.Trading)
    {
        EnemyController.Instance.enemyStatus = Random.Range(0, 2) % 2 == 0 ? EnemyController.EnemyStatus.Attacking : EnemyController.EnemyStatus.Fleeing;
    }
}
```

Figura 199: Mètode Shoot de PlayerController

El moviment de les bales de canó es basa en CannonballScript, que en el mètode Start (Figura 200) estableix la velocitat del RigidBody. La direcció depèn de la orientació del vaixell i de si es tracta d'una bala que prové de la part dreta o esquerra.

```
private void Start()
{
    rb = GetComponent<Rigidbody>();
    rb.velocity = speed * ((isRightCannonball ? 1 : -1) * shipController.transform.right + new Vector3(0, 0.2f, 0));
}
```

Figura 200: Mètode Start de CannonballScript

En el mètode Update de CannonballScript comprova si la bala s'ha enfonsat al mar sense impactar amb cap vaixell. Quan això passi s'instanciarà un ParticleSystem que simularà l'esquitx d'aigua, es reproduirà el so corresponent i quan aquest acabi es destruirà la bala. A la Figura 201 es mostra el codi que ho fa possible.

```
private void Update()
{
    if (transform.position.y <= -0.5 && !isBeingDestroyed)
    {
        isBeingDestroyed = true;
        GetComponent<SphereCollider>().enabled = false;
        StartCoroutine(CannonballMiss());
    }
}

1 referencia
public IEnumerator CannonballMiss()
{
    Instantiate(splashPrefab, transform.position, splashPrefab.transform.rotation);
    splashAudioSource.Play();
    yield return new WaitForSeconds(splashAudioSource.clip.length);
    Destroy(gameObject);
}
```

Figura 201: Mètodes Update i CannonballMiss de CannonballScript

6.7.3. Altres

El mètode Banace (Figura 202) simula el balanceig dels vaixells rotant en l'eix z de forma periòdica. Es calcula l'angle de rotació en cada fotograma amb la funció de sinus i el temps actual. Se li aplica un multiplicador per accentuar el moviment.

```
private void Balance()
{
    currentTime = (currentTime + Time.deltaTime) % (Mathf.PI * 2);
    transform.rotation = Quaternion.Euler(transform.rotation.eulerAngles.x, transform.rotation.eulerAngles.y, Mathf.Sin(currentTime) * 5f);
}
```

Figura 202: Mètode Balance

Finalment el mètode SoundAndParticles (Figura 203) gestiona l'ús de so i partícules a partir de la variable verticalInput. Aquests només seran actius quan el valor sigui superior a 0. Per evitar que s'activin i es desactivin els elements constantment s'utilitza la variable isStopped.

```
private void SoundAndParticles()
{
    if (verticalInput > 0)
    {
        if (isStopped)
        {
            isStopped = false;
            StartCoroutine(StartFade(WavesSound, 0.5f, 0.15f));
            MovingWaves.Play();
        }

    }
    else
    {
        if (!isStopped)
        {
            isStopped = true;
            StartCoroutine(StartFade(WavesSound, 1.5f, 0f));
            MovingWaves.Stop();
        }
    }
}
```

Figura 203: Mètode SoundAndParticles

El so no es reprodueix i es para de reproduir de cop, sinó que es deixa que en bucle i es fa un difuminat de volum amb la corutina StartFade (Figura 204). Per a activar i desactivar les partícules s'utilitzen els mètodes propis de ParticleSystem (Play i Stop, respectivament). A continuació es pot veure el codi.

```
public static IEnumerator StartFade(AudioSource audioSource, float duration, float targetVolume)
{
    float currentTime = 0;
    float start = audioSource.volume;
    while (currentTime < duration)
    {
        currentTime += Time.deltaTime;
        audioSource.volume = Mathf.Lerp(start, targetVolume, currentTime / duration);
        yield return null;
    }
    yield break;
}
```

Figura 204: Corutina StartFade

6.8. Gestió dels esdeveniments

En el mètode Update de GameManager es comprova la posició del vaixell del jugador i del de l'enemic i es fan modificacions en l'estat del joc conseqüentment. Per començar, el mètode HandleEnemyShipPosition calcula la distància entre els dos vaixells. Llavors, si l'enemic es troba en estat "comerciant" segueix la lògica de la Figura 205. Si està prou a prop com per comerciar, es mostra el botó. Si està una mica massa lluny, s'amaga el botó. Si en canvi està tan lluny que el jugador no el veu s'amaga fora del mapa.

```
private float HandleEnemyShipPosition(Vector3 shipPosition)
{
    Vector3 enemyShipPosition = EnemyController.Instance.enemyStatus == EnemyController.EnemyStatus.StandBy ?
        ShipScript.Instance.enemyShipTransform.position : EnemyController.Instance.transform.position;
    float distanceToEnemyShip = Vector3.Distance(shipPosition, enemyShipPosition);

    if (EnemyController.Instance.enemyStatus == EnemyController.EnemyStatus.Trading)
    {
        if (distanceToEnemyShip < distanceToInteract)
        {
            CanvasScript.Instance.CanTrade();
        }
        else if(distanceToEnemyShip < distanceToCheckIfCanInteract)
        {
            CanvasScript.Instance.CannotTrade();
        }
        else
        {
            EnemyController.Instance.HideFromMap();
        }
    }
```

Figura 205: Fragment del mètode HandleEnemyShipPosition referent a l'estat de l'enemic "comerciant"

En cas que l'enemic no estigui "comerciant", es comprova la distància amb totes les illes. Si està "a l'espera" i la distància és superior al límit establert es mourà al davant del jugador, com es mostra a la Figura 206.

```
else
{
    CanvasScript.Instance.CannotTrade();

    float minDistance = -1;
    foreach (IslandScript islandScript in islandList)
    {
        float distanceToIsland = Vector3.Distance(islandScript.transform.position, enemyShipPosition);
        if (minDistance == -1 || distanceToIsland < minDistance)
        {
            minDistance = distanceToIsland;
        }
    }

    if (EnemyController.Instance.enemyStatus == EnemyController.EnemyStatus.StandBy)
    {
        if (minDistance > distanceForEnemyToAppear)
        {
            EnemyController.Instance.transform.position = enemyShipPosition;
            EnemyController.Instance.Initialize();
        }
    }
```

Figura 206: Fragment del mètode HandleEnemyShipPosition referent a l'estat de l'enemic "a l'espera"

Llavors es gestiona la lògica de quan l'enemic està en combat, que es pot veure a la Figura 207. Si es troba en estat "lluitant" i la seva posició és molt propera a alguna illa canvia a estat fugint per abandonar el combat. En qualsevol dels dos casos, si s'allunya massa del vaixell del jugador s'amaga fora del mapa.

```
        else // El vaixell enemic està en combat
        {
            if (minDistance < distanceToDoSeaStuff)
            {
                EnemyController.Instance.enemyStatus = EnemyController.EnemyStatus.Fleeing;
            }
            if (distanceToEnemyShip > distanceToCheckIfCanInteract)
            {
                EnemyController.Instance.HideFromMap();
            }
        }

    }

    return Vector3.Distance(shipPosition, EnemyController.Instance.transform.position);
}
```

Figura 207: Fragment del mètode HandleEnemyShipPosition referent als estats de l'enemic "lluitant" i "fugint"

Seguidament es busca l'illa més propera al vaixell del jugador amb el mètode FindClosestIsland (Figura 208). Si està prou lluny de totes les illes per trobar-ne una de nova cridarà el mètode GenerateIsland. Si no n'està prou lluny però suficientment com per pescar es mostrarà el botó.

```
private void FindClosestIsland(Vector3 shipPosition, float distanceToEnemyShip)
{
    float minDistanceToIsland = -1;
    Vector3 nextIslandPosition = ShipScript.Instance.nextIslandTransform.position;
    float minDistanceToNextIsland = -1;
    IslandScript c = null;

    foreach (IslandScript islandScript in islandList)
    {
        float distanceToIsland = Vector3.Distance(islandScript.transform.position, shipPosition);
        if (c == null || distanceToIsland < minDistanceToIsland)
        {
            minDistanceToIsland = distanceToIsland;
            c = islandScript;
        }

        float distanceToNextIsland = Vector3.Distance(islandScript.transform.position, nextIslandPosition);
        if (minDistanceToNextIsland == -1 || distanceToNextIsland < minDistanceToNextIsland)
        {
            minDistanceToNextIsland = distanceToNextIsland;
        }
    }

    if (minDistanceToNextIsland > distanceBetweenIslands)
    {
        islandGenerator.GenerateIsland(new Vector2(nextIslandPosition.x, nextIslandPosition.z));
    }
    else if(minDistanceToIsland > distanceToDoSeaStuff && distanceToEnemyShip > distanceToCheckIfCanInteract)
    {
        CanvasScript.Instance.CanFish();
    }
    else
    {
        CanvasScript.Instance.CannotFish();
    }

    closestIsland = c;
}
```

Figura 208: Mètode FindClosestIsland

Finalment es busca el punt més proper a l'illa més propera amb el mètode FindClosestPointInClosestIsland (Figura 209). Es busca un punt dins la NavMesh de l'illa amb el mètode FindEntryPosition. Si està prou a prop com per deixar i recollir personatges es mostra el botó d'embarcar.

```
private void FindClosestPointInClosestIsland(Vector3 shipPosition)
{
    if (closestIsland == null || Vector3.Distance(shipPosition, closestIsland.transform.position) > distanceToCheckIfCanInteract)
    {
        PlayerIsFarFromIsland();
        return;
    }

    if (ShipScript.Instance.FindEntryPosition(distanceToInteract))
    {
        PlayerIsNearIsland();
    }
    else
    {
        PlayerIsFarFromIsland();
    }
}
```

Figura 209: Mètode FindClosestPointInClosestIsland

6.9. Sistema d'inventari

La classe InventoryScript (Figura 210) gestiona l'inventari de les illes i els vaixells. Té l'atribut inventoryCategories, que guarda tota la informació. També té les constants que indiquen la capacitat inicial que han de tenir cada una de les tres categories: "materials", "verdures" i "carns".

```
public class InventoryScript
{
    private const int materialCapacity = 100;
    private const int cropCapacity = 50;
    private const int meatCapacity = 50;
    private InventoryCategory[] inventoryCategories = null;
```

Figura 210: Classe InventoryScript

La classe InventoryCategory (Figura 211) conté les capacitats i l'ús d'inventari. La llista de recursos indica la quantitat que s'emmagatzema de cada un. La capacitat real es calcula multiplicant la capacitat base pel nivell de la categoria.

```
public class InventoryCategory
{
    public int level;
    private int baseCapacity;
    7 referencias
    public int capacity { get { return baseCapacity * level; } }
    public int usage;
    public int[] resources;

    3 referencias
    public InventoryCategory(int resourceTypeAmount)
    {
        level = 1;
        resources = new int[resourceTypeAmount];
    }

    6 referencias
    public void AddCapacity(int extraCapacity)
    {
        baseCapacity += extraCapacity;
    }
}
```

Figura 211: Classe InventoryCategories

En cas que es vulgui accedir a la informació de les categories i no s'hagi fet encara, s'inicialitzen amb el mètode InicializeResources (Figura 212).

```
private void InitializeResources()
{
    inventoryCategories = new InventoryCategory[3];
    inventoryCategories[0] = new InventoryCategory(ResourceScript.GetEnumLength(ResourceScript.ResourceType.Material));
    inventoryCategories[1] = new InventoryCategory(ResourceScript.GetEnumLength(ResourceScript.ResourceType.Crop));
    inventoryCategories[2] = new InventoryCategory(ResourceScript.GetEnumLength(ResourceScript.ResourceType.Meat));
}
```

Figura 212: Mètode InicializeResources

Quan s'afegeix un magatzem a una illa es crida el mètode AddCapacityToAllCategories (Figura 213), que suma a cada categoria la capacitat inicial respectiva.

```
public void AddCapacityToAllCategories()
{
    if (inventoryCategories == null) InitializeResources();

    inventoryCategories[0].AddCapacity(materialCapacity);
    inventoryCategories[1].AddCapacity(cropCapacity);
    inventoryCategories[2].AddCapacity(meatCapacity);
}
```

Figura 213: Mètode AddCapacityToAllCategories

En canvi, quan s'elimina un magatzem es resta la capacitat inicial amb el mètode RemoveCapacityFromAllCategories (Figura 214).

```
public void RemoveCapacityFromAllCategories()
{
    if (inventoryCategories == null) InitializeResources();

    inventoryCategories[0].AddCapacity(-materialCapacity);
    inventoryCategories[1].AddCapacity(-cropCapacity);
    inventoryCategories[2].AddCapacity(-meatCapacity);
}
```

Figura 214: Mètode RemoveCapacityFromAllCategories

Per obtenir la quantitat emmagatzemada d'un recurs particular es crida la funció GetResourceAmount (Figura 215).

```
public int GetResourceAmount(ResourceScript.ResourceType resourceType, int resourceIndex)
{
    if (inventoryCategories == null) InitializeResources();
    return inventoryCategories[(int)resourceType].resources[resourceIndex];
}
```

Figura 215: Funció GetResourceAmount

Per afegir una quantitat específica d'un recurs concret s'utilitza la funció AddResource (Figura 216). Primer es revisa si queda prou espai dins la categoria, i llavors només s'introdueix la quantitat possible.

```
public int AddResource(ResourceScript.ResourceType resourceType, int resourceIndex, int amount = 1)
{
    if (inventoryCategories == null) InitializeResources();

    int originalAmount = amount;
    InventoryCategory inventoryCategory = inventoryCategories[(int)resourceType];
    if (inventoryCategory.capacity - inventoryCategory.usage < originalAmount)
    {
        amount = inventoryCategory.capacity - inventoryCategory.usage;
    }

    inventoryCategory.resources[resourceIndex] += amount;
    inventoryCategory.usage += amount;

    return originalAmount - amount;

}
```

Figura 216: Funció AddResource de InventoryScript

Per eliminar certa quantitat d'un recurs, en canvi, s'utilitza la funció RemoveResource (Figura 217). També es comprova quina és la màxima quantitat amb la que es pot realitzar la operació.

```
public int RemoveResource(ResourceScript.ResourceType resourceType, int resourceIndex, int amount = 1)
{
    int originalAmount = amount;
    InventoryCategory inventoryCategory = inventoryCategories[(int)resourceType];
    if (GetResourceAmount(resourceType, resourceIndex) < amount)
    {
        amount = inventoryCategory.resources[resourceIndex];
    }

    inventoryCategory.resources[resourceIndex] -= amount;
    inventoryCategory.usage -= amount;

    return originalAmount - amount;
}
```

Figura 217: Funció RemoveResource

En les illes, la quantitat d'un recurs és la suma de la que es troba en els magatzems i en el vaixell del jugador, sempre que aquest estigui amarrat a l'illa. Aquest càlcul es porta a terme a la funció GetResourceAmount de IslandScript (Figura 218).

```
public int GetResourceAmount(ResourceScript.ResourceType resourceType, int resourceIndex)
{
    int amount = inventoryScript.GetResourceAmount(resourceType, resourceIndex);
    if (GameManager.Instance.isInIsland) amount += ShipScript.Instance.shipInterior.inventoryScript.GetResourceAmount(resourceType, resourceIndex);
    return amount;
}
```

Figura 218: Funció GetResourceAmount de IslandScript

Quan s'afegeix un recurs es crida el mètode AddResource de IslandScript (Figura 219). Abans de tot, si es tracta d'una verdura es comprova si en algun dels horts es necessita per poder-la plantar. Després s'intenta afegir a l'inventari de l'illa i, si no s'ha pogut introduir tot per qüestió d'espai i el vaixell està amarrat, s'intenta també en l'inventari del vaixell del jugador. Finalment, si s'ha pogut efectuar l'operació es notifica al jugador.

```
public void AddResource(ResourceScript.ResourceType resourceType, int resourceIndex, int amount = 1)
{
    int remainingAmount = amount;
    if (resourceType == ResourceScript.ResourceType.Crop)
    {
        if (!GameManager.Instance.CheckIfCropIsNew(resourceIndex))
        {
            foreach (GardenScript gardenScript in constructionList)
            {
                if (gardenScript == null) continue;

                remainingAmount = gardenScript.UseNewCrops((ResourceScript.CropType)resourceIndex, remainingAmount);

                if (remainingAmount == 0) break;
            }
        }
    }

    remainingAmount = inventoryScript.AddResource(resourceType, resourceIndex, remainingAmount);
    if (remainingAmount > 0 && GameManager.Instance.isInIsland && GameManager.Instance.closestIsland == this)
    {
        remainingAmount = ShipScript.Instance.shipInterior.inventoryScript.AddResource(resourceType, resourceIndex, remainingAmount);
    }

    if (!(GameManager.Instance.isInIsland && GameManager.Instance.closestIsland == this)) return;

    if (amount - remainingAmount > 0) CanvasScript.Instance.InventoryChange(resourceType, resourceIndex, amount - remainingAmount);
}
```

Figura 219: Mètode AddResource de IslandScript

Quan s'utilitza un recurs es crida el mètode UseResource (Figura 220), que de la mateixa manera prova d'obtenir-lo de l'inventari de l'illa i del del vaixell del jugador, sempre que sigui possible. També es notifica si s'ha pogut realitzar.

```
public bool UseResource(ResourceScript.ResourceType resourceType, int resourceIndex, int amount = 1)
{
    if (GetResourceAmount(resourceType, resourceIndex) < amount) return false;

    int remainingAmount = inventoryScript.RemoveResource(resourceType, resourceIndex, amount);
    if(remainingAmount > 0)
    {
        remainingAmount = ShipScript.Instance.shipInterior.inventoryScript.RemoveResource(resourceType, resourceIndex, remainingAmount);
    }

    if (GameManager.Instance.isInIsland && GameManager.Instance.closestIsland == this)
    {
        if (amount - remainingAmount > 0) CanvasScript.Instance.InventoryChange(resourceType, resourceIndex, -(amount - remainingAmount));
    }

    return true;
}
```

Figura 220: Mètode UseResource

6.10. Serialització

A l'hora de guardar l'estat del joc per poder recrear-lo després d'haver tancat l'escena, calia un mètode de serialització. Es va escollir JSON per la llegibilitat, el qual permet trobar errors d'escriptura molt fàcilment. Unity utilitza la llibreria JsonUtility per convertir objectes a cadenes de text i a l'inversa. Per utilitzar-la cal marcar les classes que es vulguin convertir com a serialitzables amb l'etiqueta "Serializable" a sobre de la declaració. Llavors, tots els atributs seran inclosos en la conversió, excepte els que estiguin marcats per ometre amb l'etiqueta "NonSerialized".

Malgrat això, es van trobar uns quants obstacles utilitzant aquest mètode, i tots ells han estat resolts utilitzant la llibreria Json.NET de Newtonsoft. Per començar, un camp marcat com a "no serialitzable" no apareixerà a l'inspector. Això és un problema pels valors que han de ser introduïts a través de l'editor, com corbes d'animació. Utilitzant l'etiqueta "JsonIgnore" de Newtonsoft, tal i com es mostra a la Figura 221, el camp és ignorat només en la conversió a format Json.

```
[JsonIgnore] [SerializeField] private float ageSpeed = 0.01f;
[JsonIgnore] [SerializeField] private float confortSpeed = 0.05f;
```

Figura 221: Camps amb l'etiqueta JsonIgnore

Aquesta no és una solució perfecta si s'intenta serialitzar una classe que hereta de MonoBehaviour, ja que la llibreria no accepta les propietats com Rigidbody i no es pot editar les classes pròpies de Unity per afegir l'etiqueta. En canvi, es pot canviar l'etiqueta "Serializable" de la classe per la següent cadena: "[JsonObject(MemberSerialization.OptIn)]". La opció "OptIn" comunica a la llibreria que no s'han de serialitzar els camps no marcats amb l'etiqueta "JsonProperty". Ara només cal marcar els camps que sí que es volen incloure en el resultat, com es mostra en la Figura 222.

```
[JsonObject(MemberSerialization.OptIn)]
Script de Unity (8 referencias de recurso) | 33 referencias
public class AnimalScript : MonoBehaviour
{
    [JsonProperty] public ResourceScript.AnimalType animalType;
```

Figura 222: Classe amb l'etiqueta JsonObject i atribut amb l'etiqueta JsonProperty

A més, els mètodes ToJson i FromJson de JsonUtility no són capaços de distingir entre classes germanes i només guarda els camps de la classe mare. Per solucionar-ho cal utilitzar els mètodes SerializeObject i DeserializeObject de JsonConvert, una classe de Newtonsoft. Aquests permeten incloure paràmetres de configuració per evitar errors i personalitzar el resultat. Els paràmetre "TypeNameHandling.Auto", per exemple, fa la distinció entre classes germanes. En la Figura 223 es pot veure la creació d'un objecte JsonSerializerSettings amb aquesta configuració.

```
serializerSettings = new JsonSerializerSettings
{
    TypeNameHandling = TypeNameHandling.Auto
};
```

Figura 223: Objecte JsonSerializerSettings amb la configuració TypeNameHandling.Auto

Finalment, algunes classes de Unity com els vectors i els colors tenen camps recursius, els quals fan fallar la conversió. Inicialment es va utilitzar el paràmetre de serialització "ReferenceLoopHandling.Ignore", que evita els errors de recursivitat. Més endavant es va decidir definir mètodes propis de conversió per certs tipus de camps. Es van crear classes que heretessin de JsonConverter i sobreescrivissin ReadJson i WriteJson segons les necessitats del projecte. Llavors, tot el que cal és marcar quins camps utilitzen el mètode personalitzat de conversió amb l'etiqueta de JsonConverter indicant la classe que se n'ha d'encarregar. En la Figura 224 i la Figura 225 es mostra un exemple.

```
public class ColorHandler : JsonConverter
{
    0 referencias
    public override bool CanConvert(Type objectType)
    {
        return objectType == typeof(Color);
    }

    0 referencias
    public override object ReadJson(JsonReader reader, Type objectType, object existingValue, JsonSerializer serializer)
    {
        ColorUtility.TryParseHtmlString("#" + reader.Value, out Color loadedColor);
        return loadedColor;
    }

    0 referencias
    public override void WriteJson(JsonWriter writer, object value, JsonSerializer serializer)
    {
        string val = ColorUtility.ToHtmlStringRGB((Color)value);
        writer.WriteValue(val);
    }
}
```

Figura 224: Classe ColorHandler, que hereta de JsonConverter

```
[JsonProperty] [JsonConverter(typeof(ColorHandler))] public Color _SKINCOLOR;
[JsonProperty] [JsonConverter(typeof(ColorHandler))] public Color _HAIRCOLOR;
[JsonProperty] [JsonConverter(typeof(ColorHandler))] public Color _CLOTH3COLOR;
[JsonProperty] [JsonConverter(typeof(ColorHandler))] public Color _CLOTH4COLOR;
[JsonProperty] [JsonConverter(typeof(ColorHandler))] public Color _OTHERCOLOR;
```

Figura 225: Atributs amb l'etiqueta JsonConverter

7. Resultats

7.1. Superació dels objectius

A continuació es detalla l'estat final de les tasques de desenvolupament mencionades a la introducció:

- **Navegació marítima**: L'aspecte del mar és atractiu i no consumeix excessius recursos. El sistema de controls és sòlid i fluid. El comportament del vaixell enemic respon correctament als esdeveniments. El combat és satisfactori.

- **Generació d'illes**: Les illes es creen de forma completament procedural i de forma asíncrona. S'omplen d'elements utilitzant un sistema de pooling que ajuda al rendiment. El sistema d'edició aconsegueix que les cel·les s'ajustin al relleu i permetin a l'usuari seleccionar una zona fàcilment.

- **Gestió de recursos**: El sistema d'inventari és de gran ajuda a l'hora de moure recursos. Els mètodes són intuïtius i no deixen marge per error. Es podria haver perfilat més sistema de comerç i donat una altra utilitat a les flors i bolets.

- **Control de personatges**: Els personatges es mouen de manera realista pel món. Les animacions i les fusions entre elles són creïbles. El sistema de tasques funciona, tot i que de vegades es poden produir petits errors.

- **Guardat de partida**: L'estat del joc es guarda de forma correcta en un arxiu local. El format està polit i ocupa el mínim espai possible. Hi ha un control que evita obrir arxius amb altres formats o que hagin quedat corromputs.

- **Disseny d'interfícies**: Els menús comparteixen un estil comú que casa amb la resta d'elements gràfics. La font i les icones escollides encaixen amb l'estètica. Els botons compleixen les funcions i són actius només quan toca.

- **Partícules, efectes de so i música**: Tot i que es podria haver afegit més feedback visual i auditiu per a certes accions, es considera que s'ha complert la tasca.

7.2. Legislació i normativa vigent

Aquest projecte no guarda cap tipus d'informació de caràcter personal sobre el jugador, per tant no incompleix de cap manera la LOPD (Llei Orgànica de Protecció de Dades). Tampoc no es realitza cap activitat econòmica, per tant tampoc incompleix la LSSICE (Llei de Serveis de la Societat d'Informació i Comerç Electrònic).

Pel que fa a la normativa de Copyright, tots els elements que s'han utilitzat en el joc són de creació pròpia o d'ús lliure i gratuït, per tant, no s'infringeix la normativa vigent.

7.3. PEGI

PEGI (Pan European Game Information) és un sistema de classificació d'edats europeu sobre el contingut dels videojocs. S'utilitza per classificar els jocs per edat recomanada i identifica aquells on s'utilitzen paraulotes, contingut discriminatori, de drogues, de por, d'apostes, de sexe o violent. Aquest sistema es basa en dos col·leccions d'icones: un format per 5 icones on s'especifica l'edat recomanada i un altre format per 7 icones on s'identifica el tipus de contingut. Aquestes es poden veure en la Figura 226.

Figura 226: Icones de PEGI

En el cas d'aquest projecte, només s'inclou contingut violent en forma de combat naval. Es considera, per tant, que compleix els requisits per tenir PEGI 7.

7.4. Captures

A continuació s'adjunten algunes captures del joc final.

Figura 227: Vaixell navegant

Figura 228: Mapa

Figura 229: Vaixell a punt d'amarrar

Figura 230: Anomenant illa

Figura 231: Vista de l'illa

Figura 232: Magatzem seleccionat

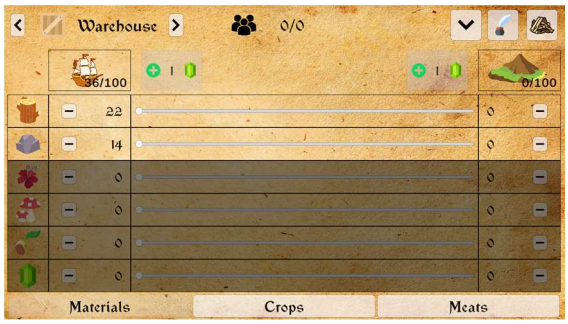


Figura 233: Menú d'inventari

Figura 234: Tripulació entrant a l'illa

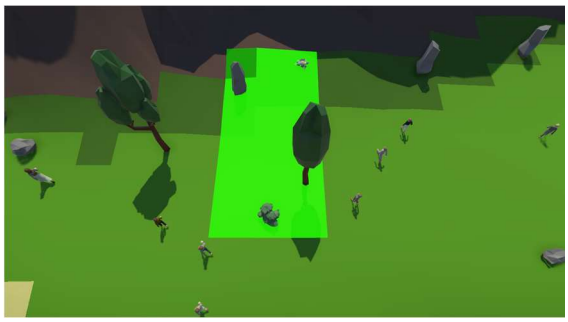


Figura 235: Seleccionant elements

Figura 236: Personatge talant un arbre

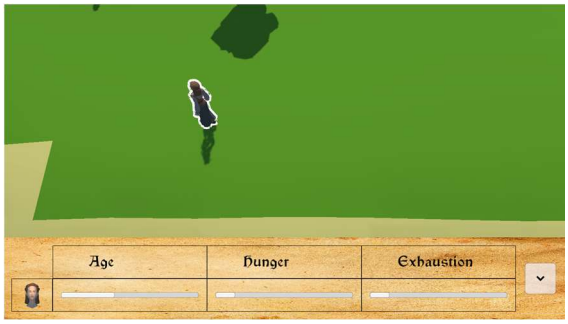


Figura 237: Detalls del personatge

Figura 238: Personatges amb gana i cansats

Figura 239: Vaixells a punt de comerciar

Figura 240: Menú d'intercanvis

Figura 241: Combat naval

Figura 242: Pantalla de fi de partida

8. Conclusions

A partir de l'estat del producte final s'ha arribat a la conclusió que s'ha aconseguit complir amb la promesa del projecte. Tot i els apartats que no han acabats fins al punt desitjat es considera que és un joc complet digne de ser llançat al mercat.

Al llarg del procés de desenvolupament d'aquest projecte s'han descobert moltes tècniques de programació que es desconeixien. Per exemple, hi ha eines de disseny d'interfícies que al principi no es consideraven necessàries però han acabat sent essencials per complir la visió original.

Això sí, també ha estat una oportunitat per aplicar els conceptes que s'han après en el grau. Les bones pràctiques inculcades en la creació de jocs seriosos han estat molt útils. Sense alguns patrons de disseny el codi hauria estat molt menys eficient. L'experiència que es va obtenir en projectes de Unity ha servit per impulsar una proposta tan ambiciosa.

Aquest projecte ha requerit molt esforç i molt temps, però s'ha obtingut satisfacció per la feina feta.

9. Futurs canvis

A continuació hi ha una llista de conceptes que no s'han arribat a desenvolupar:

- **Combat terrestre**: El paquet de models 3D de personatges que he utilitzat inclou soldats i armes, i a Mixamo es poden trobar moltes animacions de lluita cos a cos. S'hauria pogut crear un sistema de combat rudimentari, encara no estic segur de quin hauria estat el funcionament. Això permetria que una illa fos reconquerida constantment per diferents faccions i que el jugador s'hagués de preocupar de mantenir les illes sota control.

- **Escuts personalitzats**: Com que el material dels models 3D dels personatges té la imatge d'un emblema vaig pensar que seria interessant deixar que l'usuari dissenyés un escut propi a partir de certs paràmetres. Aquest apareixeria en les armadures dels soldats, en les veles del vaixell, en els cims de les illes i en la llista de partides guardades.

- **Illes especials**: Tal i com està programat el joc, les illes que van apareixent al llarg de la partida no varien massa en aspecte. Seria interessant aplicar diferents algoritmes de creació d'illes que resultés en geografia més variada.

- **Malalties i cures**: Un dels edificis que vaig acabar descartant és el del laboratori d'alquímia. El seu menú d'edició tindria un funcionament similar al sistema de pocions de Minecraft i prendria com a ingredients les flors i els bolets que es poden trobar a les illes o comprar dels comerciants. Les pocions servirien per crear remeis per les malalties de les que es contagiarien els habitants de les illes i els tripulants de la nau. A la implementació actual, les flors i els bolets només serveixen per ser venuts i aconseguir gemmes a canvi.

10. Annexes

10.1. Annex 1: Soroll en detall

En l'apartat de la creació de les illes menciono les funcions matemàtiques de generació de soroll. Hi ha molt més a explicar sobre el funcionament. A continuació explicarem de manera més detallada la informació que hem trobat al respecte.

10.1.1. Fonaments del soroll

Per generar imatges pseudoaleatòries hi ha diferents tipus de soroll, però tots es basen en el mateix principi: Tenen una permutació (Figura 243) o llista de valors que van de 0 a la llargada de la pròpia menys 1. Aquests han estat desordenats de manera que s'asseguri una bona distribució, és a dir, que no hi hagi molts valors similars seguits. Un cop desordenada, la llista es mantindrà fixa. També tenen una llavor o *seed*, que és la posició des de la qual es comença a iterar per la llista. Com que la permutació sempre serà la mateixa, cada cop que es consulti amb la mateixa llavor s'obtindrà el mateix valor.

```
int permutation[] = { 151, 160, 137, 91, 90, 15, 131, 13, 201, 95, 96, 53, 194, 233, 7, 225, 140, 36,
                      103, 30, 69, 142, 8, 99, 37, 240, 21, 10, 23, 190, 6, 148, 247, 120, 234, 75, 0,
                      26, 197, 62, 94, 252, 219, 203, 117, 35, 11, 32, 57, 177, 33, 88, 237, 149, 56,
                      87, 174, 20, 125, 136, 171, 168, 68, 175, 74, 165, 71, 134, 139, 48, 27, 166,
                      77, 146, 158, 231, 83, 111, 229, 122, 60, 211, 133, 230, 220, 105, 92, 41, 55,
                      46, 245, 40, 244, 102, 143, 54, 65, 25, 63, 161, 1, 216, 80, 73, 209, 76, 132,
                      187, 208, 89, 18, 169, 200, 196, 135, 130, 116, 188, 159, 86, 164, 100, 109,
                      198, 173, 186, 3, 64, 52, 217, 226, 250, 124, 123, 5, 202, 38, 147, 118, 126,
                      255, 82, 85, 212, 207, 206, 59, 227, 47, 16, 58, 17, 182, 189, 28, 42, 223, 183,
                      170, 213, 119, 248, 152, 2, 44, 154, 163, 70, 221, 153, 101, 155, 167, 43,
                      172, 9, 129, 22, 39, 253, 19, 98, 108, 110, 79, 113, 224, 232, 178, 185, 112,
                      104, 218, 246, 97, 228, 251, 34, 242, 193, 238, 210, 144, 12, 191, 179, 162,
                      241, 81, 51, 145, 235, 249, 14, 239, 107, 49, 192, 214, 31, 181, 199, 106,
                      157, 184, 84, 204, 176, 115, 121, 50, 45, 127, 4, 150, 254, 138, 236, 205,
                      93, 222, 114, 67, 29, 24, 72, 243, 141, 128, 195, 78, 66, 215, 61, 156, 180 };
```

Figura 243: Permutació de Perlin

La freqüència del soroll és el nombre de valors de la llista que es volen representar en el resultat. L'amplitud del soroll és l'impacte que tindran els valors en el resultat, que de moment ens podem imaginar que sempre és 1. Els valors es poden interpretar en una imatge com, per exemple, la intensitat del color. La imatge de la Figura 244, per exemple, té una freqüència de 8 i per tant es representen 8 valors.

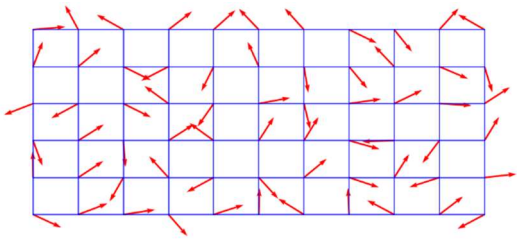
Figura 244: Imatge generada amb soroll

10.1.2. Soroll de Perlin

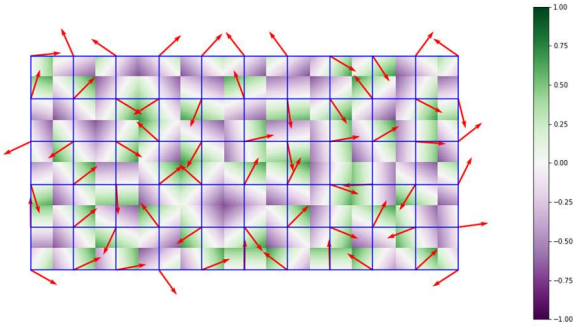
Ken Perlin va desenvolupar el soroll de Perlin, un tipus de soroll de gradients, el 1983. Per generar una imatge amb soroll Perlin es segueixen els passos següents:

- Es defineix una quadrícula de n dimensions. Estarà formada per formes geomètriques de 2^n vèrtexs. Per exemple, una quadrícula de 3 dimensions estaria formada de cubs.

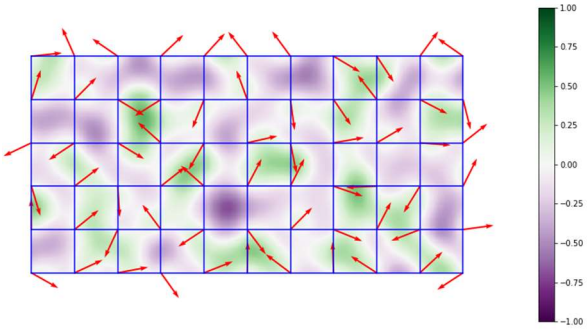
- Cada intersecció de la quadrícula tindrà associada un vector de gradient. El vector estarà normalitzat perquè tingui una longitud unitat en qualsevol dimensió. La direcció del gradient es basarà en el valor obtingut amb la permutació, la llavor i la posició de la intersecció.



- Per saber el valor que té cada punt de mostra hem de saber en quina cel·la es troba i, per tant, quins vèrtexs afecten al seu valor. Per cada un d'aquests es troba el vector "offset", que és el vector de desplaçament entre el vèrtex i el punt. Es calcula el producte escalar entre el seu vector gradient i el seu vector offset.



- Finalment, s'interpolen aquests productes escalars per unir els gradients de forma contínua. S'utilitza la segona derivada de la funció d'interpolació.



10.1.3. Soroll Símplex

La generació de soroll Símplex segueix uns passos similars al soroll de Perlin, amb algunes diferències:

- Es divideix l'espai en formes geomètriques anomenades símplex. Tenen un nombre de vèrtexs igual al nombre de dimensions més 1. Per a 1D és una línia, igual que amb el soroll de Perlin. Per a 2D és un triangle en lloc d'un quadrat, per al 3D és un tetraedre en lloc d'un cub, etc.

- Igual que amb el soroll de Perlin s'associa un vector de gradient aleatori fix a cada vèrtex i es busquen els vèrtexs de la cel·la de cada punt mostra. Enlloc del vector "offset" es calcula simplement la distància amb una funció "falloff". En 2D s'utilitza una funció de falloff radial. En 3D s'utilitza una funció de falloff esfèrica. Etc. Llavors enlloc d'un producte escalar es fa una multiplicació escalar. En la Figura 245 es mostra un exemple.

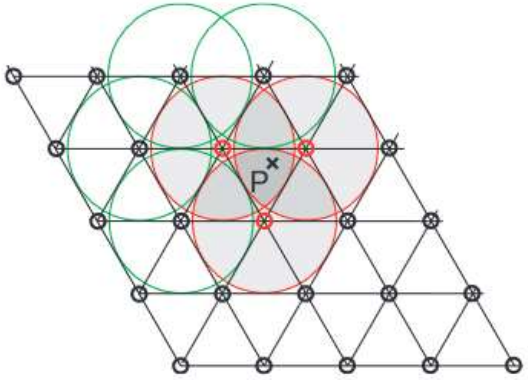


Figura 245: Càlcul de falloff radial

- Una altra diferència és en la manera de combinar els resultats. No cal fer interpolació perquè en símplex la contribució de cada punt és independent. En la Figura 246 es veu la suma de multiplicacions escalars.

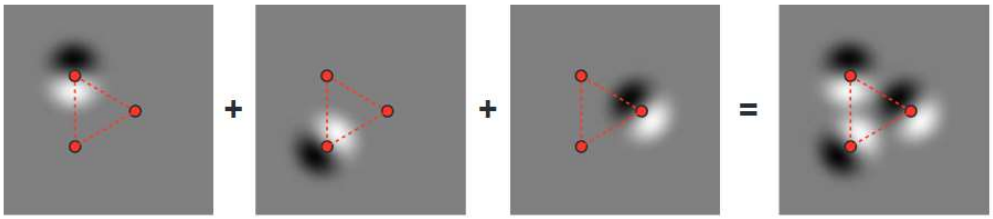


Figura 246: Combinació de multiplicacions escalars

10.1.4. Avantatges i inconvenients

El soroll Símplex té els següents avantatges respecte el soroll Perlin:

- Com que les formes geomètriques utilitzades tenen només n+1 vèrtexs enlloc de 2^n vèrtexs té una complexitat i un cost computacional menor.
- A diferència del soroll Perlin (veure Figura 247:Figura 247), la graella no és paral·lela als eixos de coordenades i no té artefactes direccionals destacables, és a dir, és visualment isotròpic.
- El seu gradient està ben definit i és continu.
- És fàcilment implementable en hardware.

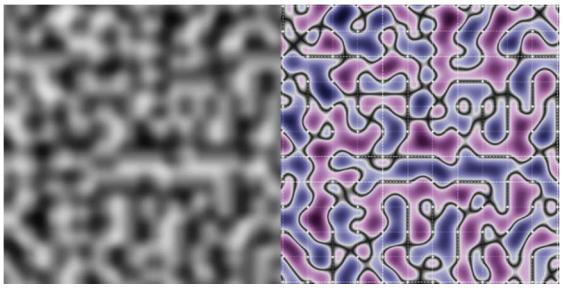


Figura 247: El soroll Perlin té artefactes direccionals

Això sí, té alguns inconvenients:

- Els sorolls generats amb la mateixa seed però per diferents dimensions són visualment diferents.
- A més, en dimensions superiors els valors obtinguts tendeixen cada cop més al centre de l'histograma i, com a resultat, sembla perdre definició (veure Figura 248).
- Com que l'ús de l'algorisme en 3 o més dimensions per generació d'imatges texturitzades va estar patentat fins el 2022 no va popularitzar-se tant com el soroll Perlin i la documentació que es pot trobar és limitada. All llarg dels anys s'han creat versions similars a l'algoritme com OpenSímplex i SuperSímplex que són lliures de patent.

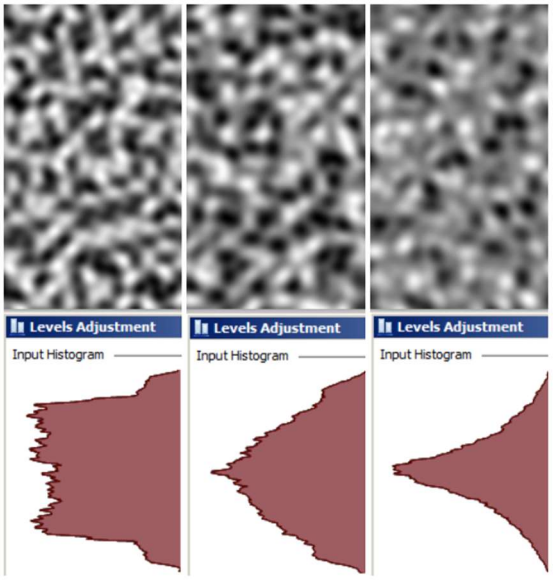


Figura 248: El soroll perd definició en dimensions superiors

10.1.5. Possibles millores del soroll

Existeixen tècniques per millorar els resultats del soroll Perlin:

- Es recomana utilitzar la versió revisada presentada l'any 2002. Aquesta comporta dues importants millores:

  El primer és canviar la funció d'interpolació bicúbica que utilitzava, anomenada "smoothstep", per una de quíntica anomenada "smootherstep". El problema que tenia la primera és que la segona derivada, que determina el pendent de la corba, era discontinua. Això significa que en les vores de les cel·les de la graella hi havia pics i valls abruptes que trencaven l'estètica. En la Figura 249 es veu la diferència entre les dues.

  El segon és definir 16 direccions fixes (Figura 250) en les quals puguin anar els gradients, enlloc de 256. Això soluciona un problema en què en alguns casos els gradients s'acumulaven d'una forma no desitjada.

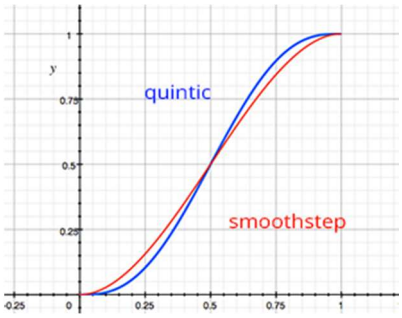


Figura 249: Diferència entre interpolació bicúbica i quíntica

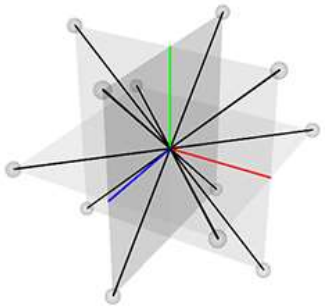


Figura 250: Les 16 direccions dels gradients

- Domain Rotation consisteix en rotar en algun eix o eixos la coordenada d'entrada del soroll, el qual fa que es diferenciï encara més el resultat de la graella definida inicialment.

- Domain Warping o Distortion consisteix en deformar el soroll aplicant una funció de distorsió com ampliar, encongir, estirar, empenyer o girar als elements. Hi ha diferents enfocaments. La tècnica tradicional consisteix en apilar diverses instàncies transformades del mateix soroll o diversos sorolls diferents, mentre que una altra implica fer alteracions en una única capa de soroll.

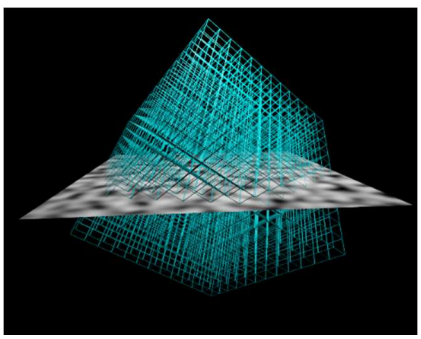


Figura 251: Domain Rotation

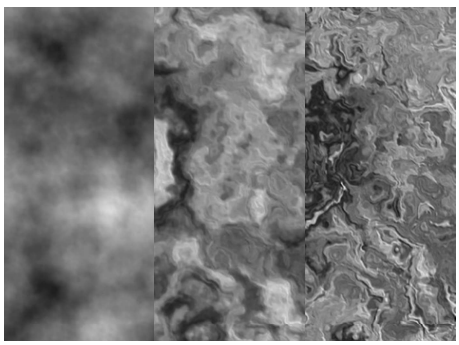


Figura 252: Domain Warping

10.2. Annex 2: Generació de colliders convexos

Tal i com explico en l'apartat de la gestió dels esdeveniments, hi ha una distància fixa amb les illes a partir de la qual el jugador tindrà l'opció d'amarrar-hi. Això es gestiona amb un simple càlcul de distància (a través del mètode Vector3.Distance) entre les posicions de les illes i el vaixell.

Aquesta aproximació no funciona en illes més grans, que tenen una forma més irregular. En alguns casos el vaixell pot trobar-se tan lluny de la costa que no veu la illa però l'opció d'amarrar és presentada de totes maneres. Per evitar-ho es va pensar en reutilitzar el mesh de l'illa per a un trigger collider escalat lleugerament per tal que detectés el jugador abans que aquest xoqués amb l'illa. Per desgràcia, el motor de físiques de Unity no permet l'ús de trigger colliders còncaus.

Afortunadament, es va trobar una llibreria (V-HACD o Voxelized Hierarchical Aproximate Convex Decomposition) que és capaç de partir un collider còncau en diversos de convexos. En la Figura 253 es veu el codi que genera els colliders convexos i els marca com a trigger, i en la Figura 254 es mostra la visualització dels diversos colliders convexos.

```
CMR.ConvexDecomposition.Bake(island, CMR.VHACDSession.Create(), null, false, true, false);
foreach (MeshCollider triggerCollider in island.transform.GetChild(0).GetComponentsInChildren<MeshCollider>())
{
    triggerCollider.isTrigger = true;
}
```

Figura 253: Codi que genera els colliders convexos i els marca com a trigger

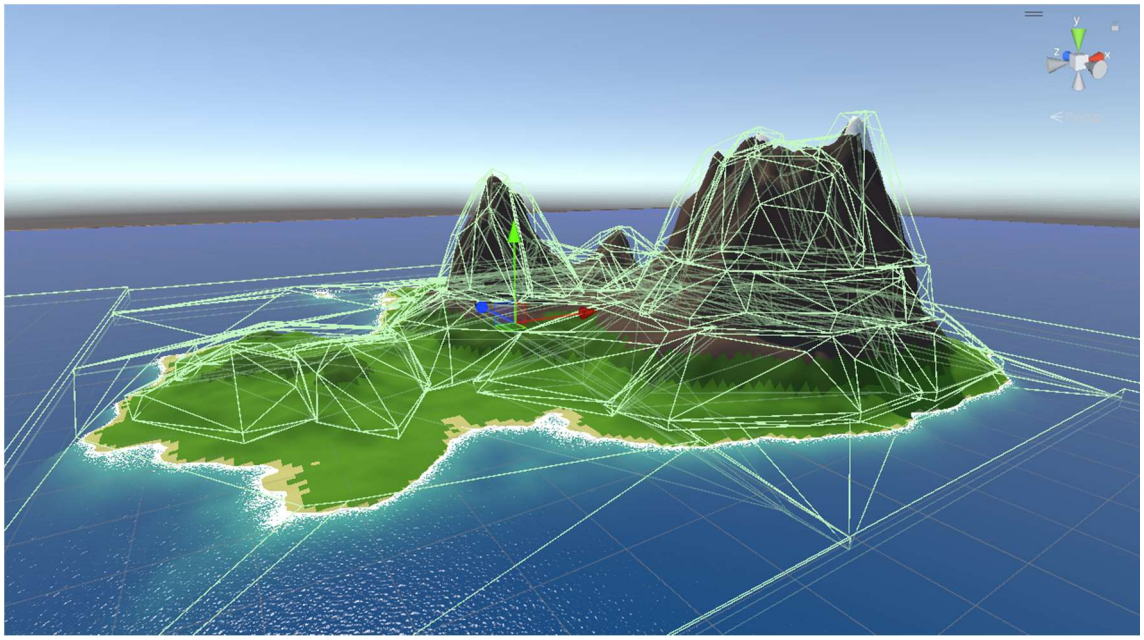


Figura 254: Visualització dels diversos colliders convexos

Llavors, per saber la distància entre el jugador i la costa d'una illa caldria iterar per cada un dels trigger colliders i trobar la distància mínima amb el mètode Physics.ClosestPoint. Si alguna d'aquestes distàncies és menor a la constant definida anteriorment, es considerarà que es pot amarrar. El codi de la Figura 255 executa aquesta lògica.

```
bool closeToIsland = false;
float minDistance = 10;
foreach(MeshCollider meshCollider in nextIsland.transform.GetChild(0).GetComponentsInChildren<MeshCollider>())
{
    Vector3 colliderClosestPoint = Physics.ClosestPoint(transform.position, meshCollider, nextIsland.transform.position, nextIsland.transform.rotation);
    float distanceToClosestPoint = Vector3.Distance(transform.position, colliderClosestPoint);
    if (distanceToClosestPoint < minDistance)
    {
        minDistance = distanceToClosestPoint;
    }
    if(distanceToClosestPoint < distanceToBoardIsland)
    {
        closeToIsland = true;
        break;
    }

}
if (closeToIsland)
{
    nextIsland.GetComponent<IslandScript>().PlayerIsNear();
}
else
{
    nextIsland.GetComponent<IslandScript>().PlayerIsFar();
}
```

Figura 255: Càlcul de la distància amb l'illa al script del jugador

Hi ha un gran problema amb l'ús d'aquests colliders, però, i és el rendiment. Per començar, es tarda uns segons a generar-los quan es crea cada illa. A més, en el pitjor dels casos (l'últim collider de la llista és el més proper) s'ha d'iterar per tots ells en cada frame per trobar la distància mínima. Finalment, interfereixen en els events del ratolí en el mode edició de l'illa i per tant s'han de desactivar i tornar a activar cada vegada que s'entra i es surt d'aquest estat, respectivament.

Com que s'ha acabat decidint crear illes més petites, en les quals el simple càlcul de distància serveix, finalment es va descartar aquest plantejament.

11. Bibliografia

- Stratos, comunidad de desarrolladores de videojuegos y multimedia. https://stratos-ad.com/
- Unity Water Scene Tutorial. YouTube. https://www.youtube.com/playlist?list=PLA6Gf0nq2Gh76TDm9mZR0loAY7KWMk4KQ
- Unity Scene With Ocean Physics and a Boat. GitHub. https://github.com/ditzel/UnityOceanWavesAndShip
- SIMPLE CARTOON WATER in Unity. YouTube. https://www.youtube.com/watch?v=Vg0L9aCRWPE
- Universal Render Pipeline in Unity 2019 – Overview & Tutorial. YouTube. https://www.youtube.com/watch?v=m6YqTrwjpP0
- Demo Project using the Universal RP from Unity3D. GitHub. https://github.com/Unity-Technologies/BoatAttack
- Boat Controller. GitHub. https://github.com/yboumaiza7/Boat-Controller
- How to fade audio in Unity: I tested every method, this one's the best. John Leonard French. https://johnleonardfrench.com/how-to-fade-audio-in-unity-i-tested-every-method-this-ones-the-best/
- Fire a Cannon in Unity 3D. Home and Learn: Games Programming Course. https://www.homeandlearn.co.uk/games-programming/unity-fire-cannon.html
- Procedural Terrain Generation. YouTube. https://youtube.com/playlist?list=PLFt_AvWsXl0eBW2EiBtl_sxmDtSgZBxB3
- Calculadora Gráfica. Desmos. https://www.desmos.com/calculator?lang=es
- Physics in Unity 5.0. Unity Documentation. https://docs.unity3d.com/Manual/UpgradeGuide5-Physics.html
- Voxelized Hierarchical Convex Decomposition - V-HACD version 4. Google Docs. https://docs.google.com/presentation/d/1OZ4mtZYrGEC8qffqb8F7Le2xzufiqvaPpRbLHKKgTIM/edit
- Animation Retargeting (Unity Tutorial). YouTube. https://www.youtube.com/watch?v=fNgPkuMgWFg
- How to use Unity NavMesh Pathfinding! (Unity Tutorial). YouTube. https://www.youtube.com/watch?v=atCOd4o7tG4

- Placing 3D objects on a Unity canvas – Tarodev. YouTube. https://www.youtube.com/watch?v=8yzpjkoE0YA
- Camera Stacking in Unity with URP! (Tutorial) – Unity. YouTube. https://www.youtube.com/watch?v=OmCjPctKkjw
- Make a Cutout Mask in Unity! (Inverted Mask) - Code Monkey. YouTube. https://www.youtube.com/watch?v=XJJl19N2KFM
- Unity Mask 3D. YouTube. https://www.youtube.com/watch?v=7vFwTt4isDY
- NavMeshBuilder.BuildNavMeshAsync() where is it? Unity Forum. https://forum.unity.com/threads/navmeshbuilder-buildnavmeshasync-where-is-it.558667/
- Asynchronous Runtime Navmesh Generation in Unity. Christopher Kempke's Blog. http://blog.chriskempke.com/blog/runtime_navmesh
- Persistent data: How to save your game states and settings. Unity Blog. https://blog.unity.com/technology/persistent-data-how-to-save-your-game-states-and-settings
- JSON Serialization. Unity Documentation. https://docs.unity3d.com/2020.1/Documentation/Manual/JSONSerialization.html
- JsonSerializationException: Self referencing loop detected. Unity Forum. https://forum.unity.com/threads/jsonserializationexception-self-referencing-loop-detected.1264253/
- JsonObjectAttribute opt-in serialization. Newtonsoft. https://www.newtonsoft.com/json/help/html/JsonObjectAttributeOptIn.htm
- Serialize & Deserialize Color objects in Unity - Medium. https://medium.com/@altaf.navalur/serialize-deserialize-color-objects-in-unity-1731e580af94
- How to Create a Custom JsonConverter in Json.NET. Code Maze. https://code-maze.com/json-dotnet-create-custom-jsonconverter/
- Free APA Citation Generator. Scribbr. https://www.scribbr.com/apa-citation-generator/

12. Recursos utilitzats

12.1. Models i animacions

12.1.1. Construccions

Low Poly Ship. CGTrader.
https://www.cgtrader.com/free-3d-models/watercraft/military-watercraft/low-poly-ship-f1e4fef1-c2b9-4ca7-ac5f-352fe1bfef16

Free Elf House 3D Low Poly Pack. CraftPix.Net.
https://craftpix.net/freebies/free-elf-house-3d-low-poly-models/

Gold mine. CGTrader.
https://www.cgtrader.com/free-3d-models/various/various-models/gold-mine-f5da52e5-1b75-4f4f-af17-033fc6c53dff

Low Poly Fence Kit. Sketchfab.
https://sketchfab.com/3d-models/low-poly-fence-kit-8e3c6de030424076b80772453877c494

12.1.2. Personatges i animals

Lowpoly Medieval Peasants. Unity Asset Store.
https://assetstore.unity.com/packages/3d/characters/humanoids/humans/lowpoly-medieval-peasants-free-pack-122225

Low Poly Farm Animals 3d model. Creazilla.
https://creazilla.com/nodes/5228-low-poly-farm-animals-3d-model

Chicken – rigged. Sketchfab.
https://sketchfab.com/3d-models/chicken-rigged-6e3b93c078114c52bfe4cfa08b9843eb

Free Farming Crops 3D Low Poly Models. CraftPix.Net.
https://craftpix.net/freebies/free-farming-crops-3d-low-poly-models/

Free Medieval Props 3D Low Poly Pack. CraftPix.Net.
https://craftpix.net/freebies/free-medieval-props-3d-low-poly-pack/

Watering Can. Sketchfab.
https://sketchfab.com/3d-models/watering-can-ffb0d644238b4c679658aa0ee46ac6da

Low poly compass (3 pieces). Sketchfab.
https://sketchfab.com/3d-models/low-poly-compass-3-pieces-c6e6060d8de64c78bd1be88e6077c4b4

Mixamo.
https://www.mixamo.com/

12.1.3. Elements de l'illa

Free Bush 3D Low Poly Pack. CraftPix.Net.
https://craftpix.net/freebies/free-bush-3d-low-poly-models/

Free Tree 3D Low Poly Pack. CraftPix.Net.
https://craftpix.net/freebies/free-tree-3d-low-poly-pack/

Free Stone 3D Low Poly Pack. CraftPix.Net.
https://craftpix.net/freebies/free-stone-3d-low-poly-models/

Free Shrubs, Flowers and Mushrooms 3D Low Poly Models. CraftPix.Net.
https://craftpix.net/freebies/free-shrubs-flowers-and-mushrooms-3d-low-poly-models/

12.2. Imatges

12.2.1. Mapa

Old cartoon ship drawing with white sails. iStock.
https://www.istockphoto.com/es/vector/historieta-barco-de-vela-gm870971950-145480005

Vector mountain landscape illustration. iStock.
https://www.istockphoto.com/es/vector/vector-ilustraci%C3%B3n-a-las-monta%C3%B1as-gm503242471-44209966

Old blank map background. Adobe Stock.
https://stock.adobe.com/es/images/old-blank-map-background/133265773

Pin Icon - 2377922. Flaticon.
https://www.flaticon.com/free-icon/pin_2377922

Paper texture background, old yellow design. Rawpixel.
https://www.rawpixel.com/image/6129082/paper-texture-background-old-yellow-design

Vintage wind rose symbol, ancient compass icon on old paper. iStock.
https://www.istockphoto.com/es/vector/s%C3%ADmbolo-de-rosa-de-viento-vintage-icono-de-la-br%C3%BAjula-antigua-en-papel-amarillo-gm1210297571-350565590

12.2.2. Materials

Wood Icon - 2153144. Flaticon.
https://www.flaticon.com/free-icon/wood_2153144

Granite Icon - 6223796. Flaticon.
https://www.flaticon.com/free-icon/granite_6223796

Gem Icon - 2778273. Flaticon.
https://www.flaticon.com/free-icon/gem_2778273

Hibiscus Icon - 5130984. Flaticon.
https://www.flaticon.com/free-icon/hibiscus_5130984

Mushroom Icon - 2659432. Flaticon.
https://www.flaticon.com/free-icon/mushroom_2659432

12.2.3. Verdures

Onion Icon - 1330443. Flaticon.
https://www.flaticon.com/free-icon/onion_1330443

Carrot Icon - 6108769. Flaticon.
https://www.flaticon.com/free-icon/carrot_6108769

Eggplant Icon - 8775308. Flaticon.
https://www.flaticon.com/free-icon/eggplant_8775308

Cucumber Icon - 1515011. Flaticon.
https://www.flaticon.com/free-icon/cucumber_1515011

Cabbage Icon - 4056895. Flaticon.
https://www.flaticon.com/free-icon/cabbage_4056895

Potato Icon - 1652077. Flaticon.
https://www.flaticon.com/free-icon/potato_1652077

Tomato Icon - 5948368. Flaticon.
https://www.flaticon.com/free-icon/tomato_5948368

Zucchini Icon - 6108510. Flaticon.
https://www.flaticon.com/free-icon/zucchini_6108510

Bell pepper Icon - 5346673. Flaticon.
https://www.flaticon.com/free-icon/bell-pepper_5346673

Corn Icon - 895140. Flaticon.
https://www.flaticon.com/free-icon/corn_895140

12.2.4. Animals

Cow Icon - 3969788. Flaticon.
https://www.flaticon.com/free-icon/cow_3969788

Cow Icon - 3281151. Flaticon.
https://www.flaticon.com/free-icon/cow_3281151

Pig Icon - 3969799. Flaticon.
https://www.flaticon.com/free-icon/pig_3969799

Pig Icon - 3281199. Flaticon.
https://www.flaticon.com/free-icon/pig_3281199

Lamb Icon - 1592623. Flaticon.
https://www.flaticon.com/free-icon/lamb_1592623

Sheep Icon - 3969801. Flaticon.
https://www.flaticon.com/free-icon/sheep_3969801

Chicken Icon - 347439. Flaticon.
https://www.flaticon.com/free-icon/chicken_347439

Chicken Icon - 3969783. Flaticon.
https://www.flaticon.com/free-icon/chicken_3969783

12.2.5. Carn

Steak Icon - 3005159. Flaticon.
https://www.flaticon.com/free-icon/steak_3005159

Pork Icon - 3005118. Flaticon.
https://www.flaticon.com/free-icon/pork_3005118

Lamb Icon - 3005132. Flaticon.
https://www.flaticon.com/free-icon/lamb_3005132

Drumstick Icon - 6233197. Flaticon.
https://www.flaticon.com/free-icon/drumstick_6233197

Fish Icon - 1979686. Flaticon.
https://www.flaticon.com/free-icon/fish_1979686

Steak Icon - 3005130. Flaticon.
https://www.flaticon.com/free-icon/steak_3005130

Chicken Icon - 4391772. Flaticon.
https://www.flaticon.com/free-icon/chicken_4391772

Fish Icon - 1028182. Flaticon.
https://www.flaticon.com/free-icon/fish_1028182

12.2.6. Altres

Multiple Users Silhouette Icon - 33308. Flaticon.
https://www.flaticon.com/free-icon/multiple-users-silhouette_33308

Vectorwin. Pile wood timber color icon vector image. VectorStock.
https://www.vectorstock.com/royalty-free-vector/pile-wood-timber-color-icon-vector-43650060

Mayflower Ship Icon - 8823359. Flaticon.
https://www.flaticon.com/free-icon/mayflower-ship_8823359

Crate Icon - 3017266. Flaticon.
https://www.flaticon.com/free-icon/crate_3017266

Beer Icon - 3884804. Flaticon.
https://www.flaticon.com/free-icon/beer_3884804

Night Icon - 10291128. Flaticon.
https://www.flaticon.com/free-icon/night_10291128

Vegetable Icon - 6265882. Flaticon.
https://www.flaticon.com/free-icon/vegetable_6265882

Livestock Icon - 5312800. Flaticon.
https://www.flaticon.com/free-icon/livestock_5312800

Pickaxe Icon - 2736385. Flaticon.
https://www.flaticon.com/free-icon/pickaxe_2736385

Plant Icon - 2303716. Flaticon.
https://www.flaticon.com/free-icon/plant_2303716

Stump Icon - 4951128. Flaticon.
https://www.flaticon.com/free-icon/stump_4951128

Block Icon - 7596657. Flaticon.
https://www.flaticon.com/free-icon/block_7596657

Bowl Icon - 7284726. Flaticon.
https://www.flaticon.com/free-icon/bowl_7284726

Sunburst. FreePNG.es.
https://www.freepng.es/png-1psvyl/

Vinque Font Family. 1001 Fonts.
https://www.1001fonts.com/vinque-font.html

Plus Icon - 4315614. Flaticon.
https://www.flaticon.com/free-icon/plus_4315614

Minus Icon - 4315587. Flaticon.
https://www.flaticon.com/free-icon/minus-button_4315587

Inkwell Icon - 903311. Flaticon.
https://www.flaticon.com/free-icon/inkwell_903311

Hammer Icon - 9507920. Flaticon.
https://www.flaticon.com/free-icon/hammer_9507920

Left Chevron Icon - 1633718. Flaticon.
https://www.flaticon.com/free-icon/left-chevron_1633718

Dice Icon - 7262385. Flaticon.
https://www.flaticon.com/free-icon/dice_7262385

Paper Coloring book Drawing Black and white, Crumpled paper, angle, white png. PNGEgg.
https://www.pngegg.com/en/png-doxyv

Old parchment paper scroll sheet vintage aged or texture background. Vecteezy.
https://www.vecteezy.com/png/12981783-old-parchment-paper-scroll-sheet-vintage-aged-or-texture-background

BabySofja. (n.d.). Seamless pattern ground with stones, brown soil texture. iStock.
https://www.istockphoto.com/es/vector/terreno-de-patr%C3%B3n-sin-costuras-con-piedras-textura-de-suelo-marr%C3%B3n-para-papel-gm1313969385-402324834

12.3. Efectes de so

Sailing Ship, by Michael Ghelfi. Michaël Ghelfi Studios.
https://michaelghelfi.bandcamp.com/track/sailing-ship-4

Three Masted Schooner, On Board: Bow Cutting Through Water. YouTube.
https://www.youtube.com/watch?v=R7u0rOsmMgg

Sound Ideas - Topic. 40 Mm Cannon Firing 5. YouTube.
https://www.youtube.com/watch?v=TkDw3D2HnCM&list=OLAK5uy_nyeEPrspYD_oK7T1Mi51LWedTSlkpuKLg&index=75

Sound Library. All Wood Explosion Debris Sounds (Fortnite) - Sound Effect for editing. YouTube.
https://www.youtube.com/watch?v=3UBUQEqAOiQ

Sounds of a Lifetime. Big Water Splash Sound Effect. YouTube.
https://www.youtube.com/watch?v=xHN3zSp6Ggg

Sound Effect Database. Heavy Chain Sound Effect. YouTube.
https://www.youtube.com/watch?v=EiyYdyRALgI

Best sounds. Water pouring - Sound Effect (HD). YouTube.
https://www.youtube.com/watch?v=2PZJMbBwo9g

Sound Library. Ship Bell - Sound Effect for editing. YouTube.
https://www.youtube.com/watch?v=nBlOblwqce4

Fishing Reel Sound Effect Fx [Video]. YouTube.
https://www.youtube.com/watch?v=MUny9Nl27Zs

Page Flip Sound Effect No copyrhgt . . .sound effect LNC [Video]. YouTube. https://www.youtube.com/watch?v=MpFMIEk9IV0

Minimal UI Sounds. Unity Asset Store. https://assetstore.unity.com/packages/audio/sound-fx/minimal-ui-sounds-78266

12.4. Música

Royalty Free Medieval Music - "Marked" by Alexander Nakarada. Youtube. https://www.youtube.com/watch?v=4BRcY4o06A0

Royalty Free Medieval Music - "Celebration" by Alexander Nakarada. Youtube. https://www.youtube.com/watch?v=h-aqStDuObU

Royalty Free Medieval Music - " Entertainment" by Alexander Nakarada. Youtube. https://www.youtube.com/watch?v=hOGLCoYangU

Royalty Free Melodic Celtic Fantasy Music "The Road Home", by Alexander Nakarada. YouTube. https://www.youtube.com/watch?v=Gxg9VF3m5lg

Medieval Song Village Consort [No Copyright Music]. YouTube. https://www.youtube.com/watch?v=eZ_r1H9vHkI

12.5. Altres

Stylized Water For URP | VFX Shaders. Unity Asset Store. https://assetstore.unity.com/packages/vfx/shaders/stylized-water-for-urp-162025

Collision Mesh Generator | Convex Decomposition. Unity Asset Collection. https://unityassetcollection.com/collision-mesh-generator-convex-decomposition-free-download/

Quick Outline - Unity Asset Store. https://assetstore.unity.com/packages/tools/particles-effects/quick-outline-115488

13. Manual d'usuari i d'instal·lació

En el següent enllaç es pot trobar el repositori de Github, on està penjat el projecte de Unity: https://github.com/kimarsal/Colono

Per obtenir el contingut del repositori només cal clonar-lo a través de Git o descarregar-lo com un arxiu comprimit. En la Figura 256 es mostren les opcions que apareixen a la pàgina web.

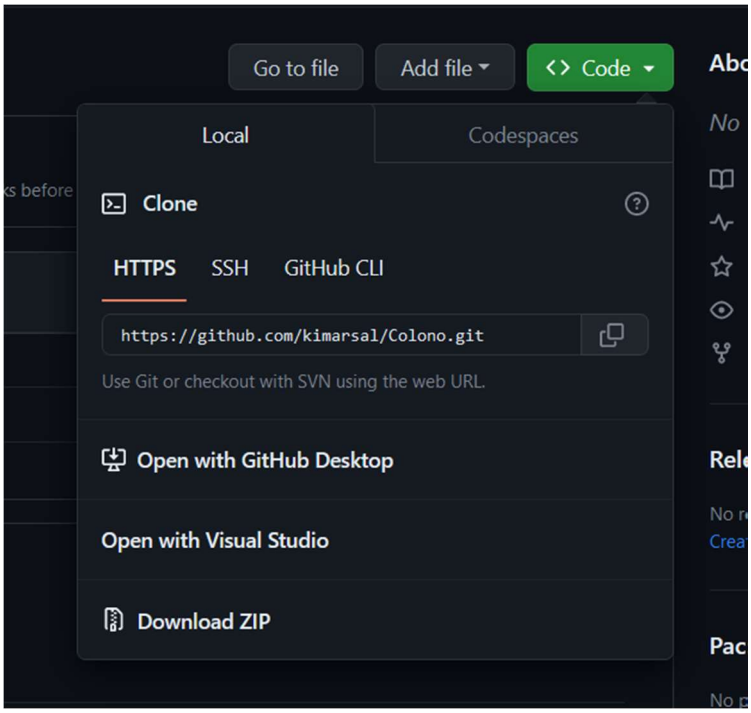


Figura 256: Opcions de descàrrega del repositori

El joc està disponible per a ordinador. El compilat es pot trobar en el següent enllaç: https://drive.google.com/file/d/1p6kxb9aLkMl569xeicF9fgyp4qJvxxDb/view?usp=sharing

Un cop estigui descarregat es pot obrir el joc seleccionant l'executable "Colono.exe". Els controls del joc són els següents:

- **Ratolí**: Interactuar amb l'entorn i la UI.
- **Tecles WASD o fletxes**: Moviment del vaixell o de la càmera.
- **Tecla espai**: Disparar bales de canó.